Java Reference – Language

By John Bentley

Table of Contents

Key

This is text is for general narration.
The next paragraph.

This is a rule to follow.

```
This contains any code or other examples.
```

This justifies the rule.

> This is a quote that also justifies the rule.

This is the source for the rule (if there are not multiple source that refer to different parts of the example, justification, or justification quote).

Another rule to follow.

Start

Hello World

Write source file. Create a text file with the code below and save as HelloWorld.java.

```
// HelloWorld.java (Source code)
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Compile into bytecode.

```
C:\dir>javac HelloWorld.java
// Outputs …HelloWorld.class (Compiled "Bytecode)
```

Run the program.

```
C:\dir>java HelloWorld
```

It is the bytecode, in HelloWorld.class, that runs in a Java Virtual Machine.

The Java Platform API

The Java Platform API is the framework

http://download.oracle.com/javase/6/docs/api/index.html
http://download.oracle.com/javase/7/docs/api/index.html

Java File and Directory setup

Source and Bytecode File Locations

The convention used by IntelliJ is as follows.

```
// Source file
<ProjectRoot>\src\com\example\graphics\ProjectName.java

// Bytecode
<ProjectRoot>\out\production\ProgjectName\com\example\graphics\ProjectName.class

// Where \com\example is the reverse internet domain name of the organisation, \graphics is
the name of the package
```

This structure allows you to hand off byte code class files without revealing your source code.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/package/managingfiles.html

CLASSPATH

General

The JVM looks for your class files by searching:

- The current directory;

- The JAR file containing the Java platform classes;
- The directory set by the CLASSPATH environment variable;

The CLASSPATH variable is one way to tell applications, including the JDK tools, where to look for user classes.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/environment/paths.html

When the JVM uses the CLASSPATH environment variable it finds your class files by adding the package name to the CLASSPATH.

```
// CLASSPATH
<ProjectRoot>\bin\

// Package name
com.example.graphics

// JVM looks for .class files in
<ProjectRoot>\bin\com\example\graphics
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/package/managingfiles.html

To inform the Java Virtual Machine (JVM) of the CLASSPATH (bin) directory you can do this in two ways:

- Set the CLASSPATH environment variable at the windows level;
- Set an IDE CLASSPATH.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/package/managingfiles.html

Windows command for CLASSPATH

```
// Display DOS
C:\> set CLASSPATH
C:\> echo %CLASSPATH%

// Display Powershell
C:\> Get-Item Env:\CLASSPATH

// Delete the current contents of CLASSPATH
C:\> set CLASSPATH=

// Set
C:\> set CLASSPATH=C:\Users\John\Sda\Code\Java\Examples\TutorialAtOracle\HelloWorld\bin

// Set as environment variable (not recommended)
// To Set in Windows
Control Panel > System > Advanced System Settings > Advanced > Environment Variables ... >
System Variables > CLASSPATH
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/package/managingfiles.html

IDE dependency setting in lieu of adding a CLASSPATH

In lieu of setting a CLASSPATH variable in IntelliJ IDEA, on a project basis, setup a dependency as follows:

- {IntelliJ IDEA} > File > Project Structure...
- |Project Settings| > |Modules| > MyModule select.
- Add either kind of dependency:
    - Add a Jar or Directory

        - |Dependencies| > [+] > [Jars or Directories ...]
        - Navigate to your dependency, e.g. "C:\Program Files (x86)\MySQL\Connector J 8.0\mysql-connector-java-8.0.11.jar". [OK]
        - Observe the dependency has been listed.
    - Add a Java Library
        - |Dependencies| > [+] > [Library ...] > [Java]
        - Navigate to your dependency, e.g. "C:\Program Files (x86)\MySQL\Connector J 8.0\mysql-connector-java-8.0.11.jar". [OK]
        - Observe the dependency has been listed.

Run console applications created by IntelliJ IDEA

From within the IDE

- IntelliJ > Run > Edit Configurations …
- IntelliJ > Run > Run …

From a command line

By reference to main class

To run a console application from the windows command line (e.g. from powershell), by reference to the main class.

```
PS ...\Sda\Code\Java\Apps\BiblatexSort\out\production\BiblatexSort> java
au.com.softmake.BiblatexSort
```

Assuming:

- "BiblatexSort" is the name of the public class with a main method (which must be the name of the java file);
- "au.com.softmake.BiblatexSort" is the root package name.
- Apps\BiblatexSort\out\production\BiblatexSort\au\com\softmake\BiblatexSort.class is the bytecode we are attempting to run.

(Bentley, C:\Users\John\Documents\Sda\Code\Java\Apps\BiblatexSort)

Also can try, if Classpath has been set and conflicts, ...

```
PS ...\Sda\Code\Java\Apps\BiblatexSort\out\production\BiblatexSort> java -classpath .
au.com.softmake.BiblatexSort
```

By reference to jar file

To run a console application from the windows command line (e.g. from powershell), by reference to a jar file.

- Compile a Jar file:
    - IntelliJ > File > Project Structure … > Project Settings > Artificats > [+] > JAR > From module with dependencies …
    - |Create Jar from Modules|
        - Main Class: […] > (Choose the main class. e.g. au.com.softmake.BiblatexSort) > [OK]
        - Jar files from libraries: extract to the target Jar.

        - Directory for the META-INF/MANIFEST.MF: C:\Users\John\Documents\Sda\Code\Java\Apps\BiblatexSort\src
        - [OK] > [OK]
    - IntelliJ > Build > Build Artifacts > BiblatexSort:jar > [Build]
- Run the Jar file:
    - Open a powershell. Run the following, being sur to use the `-jar` switch.

```
PS> Set-Location
C:\Users\John\Documents\Sda\Code\Java\Apps\BiblatexSort\out\artifacts\BiblatexSor
t_jar
// In this particular case ensure a resource file is available in the same
directory, e.g. ZoteroBiblatexExport.bib
PS> java -jar BiblatexSort.jar
```

(Bentley Experiment, 2017-12-05, C:\Users\John\Documents\Sda\Code\Java\Apps\BiblatexSort)

The Platform Environment

An application runs in a *platform environment.*

A platform environment entails the operating system, java virtual machine, class libraries, and configuration data.

Configuration Utilities

Application Properties

You can use java.util.Properties to handle your application settings.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/environment/properties.html

java.util.Properties is a key/value pair that inherits from java.util.Hashtable.

Properties example

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Properties;

public class AppProperties {
  final String appPropertiesFileName = "appProperties.txt";
  Properties appProperties;

  public AppProperties() throws IOException {
    try {
      FileInputStream fileInputStream = new FileInputStream(appPropertiesFileName);

      this.appProperties = new Properties();
      this.appProperties.load(fileInputStream);
      fileInputStream.close();

    } catch (FileNotFoundException e) {
      System.err.println("Please create " + appPropertiesFileName
          + "in the root of your application directory.");
    }
  }

  public String getProperty(String key) {
    return this.appProperties.getProperty(key);
  }

  public void setProperty(String key, String value) {
    this.appProperties.setProperty(key, value);
  }

  public void close() throws IOException {
    FileOutputStream fileOutputStream = new FileOutputStream(appPropertiesFileName);
    appProperties.store(fileOutputStream, null);
    fileOutputStream.close();
  }
}

...

public static void start() throws IOException {
  AppProperties appProperties = new AppProperties();

  System.out.println(appProperties.getProperty("Key02"));

  appProperties.setProperty("Key01", "Banana");
  appProperties.setProperty("Key02", "Apple");
  appProperties.setProperty("Key03", "Orange");

  appProperties.close();

  System.out.println("PropertiesDemo Done");
}
```

Command line arguments

Handle command line arguments.

```
// HelloWorld.java
class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
    for (String argument : args) {
      System.out.println(argument);
    }
  }
}

>java HelloWorld apple orange
Hello World!
apple
orange
```

To parse numeric arguments use parseXXX of a number class (Integer, Float, Double, etc).

```
int firstArg = Integer.parseInt(args[0]);
```

Test for no arguments.

```
if (args.length == 0) {
    System.out.println("No arguments supplied");
    return;
}
```

See also Arrays > Array Operations > Test

Supply arguments at runtime.

```
static public void main(String[] args) throws Exception {

    String xmlFileName = "";
    String xmlPathSlashFileName = "";

    String txtFileName = "";
    String xsltFileName = "";
    String xsdFileName = "";

    if (args.length == 0) {
        // Note you can override any of the following by supplying
        // at least one argument in a run configuration (Run > Edit Configurations ...)

        System.out.println("No arguments supplied via Run configuration, we'll create some
in main()");
        /**
         * Ensure Run > Edit Configurations ... is something like
         * Name: JaxpDemo args supplied in Main
         * Main Class: JaxpDemo
         * Working Directory:
C:\Users\John\Documents\Sda\Code\BjaxXsl\Examples\AviationWeatherJava\XmlSource
         *
         * ... Then set arguments here in order to switch quickly between combinations
         */
        args = new String[] {"-apiType:xsl", "-readType:readXml",
"takeOffAndLandingWeatherReports.xml", "nameValuePairs.xslt"};
    }
   ...
}
```

C:\Users\John\Documents\Sda\Code\BjaxXsl\Examples\AviationWeatherJava\JaxpDemo\src\JaxpDemo.java

Environment Variables

Loop through environment variables.

```
import java.util.Map;

private static void printEnvironmentVariables() {
  Map<String, String> environmentMap = System.getenv();

  // Loop through environment variables
  for (String environmentVariableName : environmentMap.keySet()) {
    System.out.format("%s = %s%n", environmentVariableName,
        environmentMap.get(environmentVariableName));
  }
}

// Output
USERPROFILE = C:\Users\John
ProgramData = C:\ProgramData
PATHEXT = .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
...
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) EnvironmentVariablesDemo

Reference a particular environment variable.

```
private static void printEnvironmentVariable(String envKey) {
  Map<String, String> environmentMap = System.getenv();
  System.out.printf("%s = %s", envKey, environmentMap.get(envKey));
}

printEnvironmentVariable("LOCALAPPDATA");

// Output
LOCALAPPDATA = C:\Users\John\AppData\Local
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) EnvironmentVariablesDemo

Other Configuration Utilities

Other configuration utilities.

Utility	**Description**	**Reference**

Preferences API	Store and retrieve config data in a backing store.	Preferences API Guide.
JAR archive and manifest		Packaging Programs in JAR Files
Service Provider Facility	A service provider is an implementation of a *service* — a well-known set of interfaces and (usually abstract) classes. The classes in a service provider typically implement the interfaces and subclass the classes defined in the service.	`java.util.ServiceLoader`, The Extension Mechanism

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/environment/other.html

System Utilities

System Properties

Output specific system properties

```
System.out.printf("%s = %s", java.version, System.getProperty(java.version));
System.out.printf("%s = %s", user.country, System.getProperty(user.country));

// Output
java.version = 1.6.0_37
user.country = AU
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) SystemUtilitiesDemo

List all system properties.

```
System.getProperties().list(System.out);

// Output
-- listing properties --
java.runtime.name=Java(TM) SE Runtime Environment
sun.boot.library.path=C:\Program Files\Java\jre6\bin
java.vm.version=20.12-b01
...
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) SystemUtilitiesDemo

Security Manager

When apps run in various contexts (e.g. a web applet) they may be subject to restrictions by the *security manager*. Otherwise the app can do anything. If there is a *security manager* then use as follows.

```
// Invoke
SecurityManager appsm = System.getSecurityManager();

// Check permissions with checkXXX()
SecurityManager.checkAccess  // Check thread access
SecurityManager.checkPropertyAccess
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/environment/security.html

If you don't have permissions then a SecurityException, or a subclass of SecurityException, will be thrown.

Miscellaneous System methods

System.arraycopy()

See Array Operations

Timing methods. Invoke the methods twice and subtract the difference for the elapsed time. Not necessarily accurate. Don't use to get current time (use java.util.Calendar.getInstance istead).

```
// Milliseconds
System.currentTimeMillis();

// Nanoseconds
System.nanoTime();
```

Terminate the app.

```
// Normal termination: By convention use zero.
System.exit(0);

// Terminate with error: By convention use any non-zero code.
System.exit(-1);
```

PATH and CLASSPATH

Set Windows Environment as follows:

```
JAVA_HOME : C:\Program Files\Java\jdk1.8.0_112
JDK_HOME  : %JAVA_HOME%
JRE_HOME  : %JAVA_HOME%\jre
CLASSPATH : .;%JAVA_HOME%\lib;%JAVA_HOME%\jre\lib
PATH      : your-unique-entries;%JAVA_HOME%\bin (make sure that the longish your-unique-
entries does not contain any other references to another Java installation folder.

When JDK is installed, it adds to the system environment variable Path an entry
C:\ProgramData\Oracle\Java\javapath;. I anecdotally noticed that the links in that directory
didn't get updated during an JDK installation update. So it's best to remove
C:\ProgramData\Oracle\Java\javapath; from the Path system environment variable in order to
have a consistent environment.
```

https://stackoverflow.com/questions/1672281/environment-variables-for-java-installation/26640589#26640589

PATH

The Path environment variable defines where the bin folder of the JDK is located. Setting it in the OS (e.g. windows) is optional but recommended as a convenience method for running JDK binaries from the command line without having to specify the full path to the binary every time.

```
// To Set in Windows
Control Panel > System > Advanced System Settings > Advanced > Environment Variables ... >
System Variables > Path

// You can also set the Path by also setting a JAVA_HOME System variable
JAVA_HOME=C:\Program Files\Java\jdk1.6.0_37
PATH=...;%JAVA_HOME%\bin;
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/environment/paths.html

To verify JAVA_HOME properly set.

```
// From a command prompt
>echo %JAVA_HOME%

// From a powershell prompt
PS>$env:JAVA_HOME

// Should return: "C:\Program Files\Java\jdkX.X.X_XX"
```

To verify the Path is properly set.

```
// From any folder (not C:\Program Files\Java\jdkX.X.X_XX\bin) command window
>java -version

// If it returns interesting info: you are good.
```

CLASSPATH

See CLASSPATH, Page 7, above.
Further info also at (Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/environment/paths.html

Add a repository library

Via Maven

1. You want to add, for example, ...

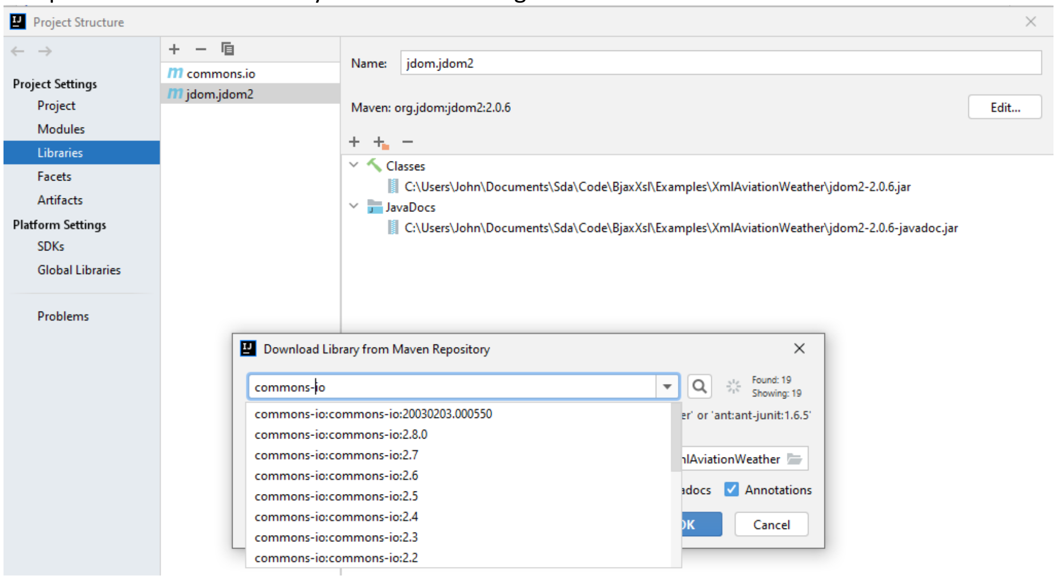
   ```
   import org.apache.commons.io.FilenameUtils;
   ```

2. Search for the name of your library as on the maven website.

   ```
   https://mvnrepository.com/artifact/org.apache.commons

   Observe under "Commons IO"
   org.apache.commons » commons-io

   Your name is "commons-io"
   ```

3. IntelliJ > File > Project Structure ... > Project Settings > Libraries > + > From Maven ...

4. In the search box enter "commons-io" (note the hypen). From the results in the drop down box choose your version e.g. "commons-io:commons-io:2.11.0" ...



Project Structure

Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems

commons.io
jdom.jdom2

Name: jdom.jdom2

Maven: org.jdom:jdom2:2.0.6

Classes
C:\Users\John\Documents\Sda\Code\BjaxXsl\Examples\XmlAviationWeather\jdom2-2.0.6.jar
JavaDocs
C:\Users\John\Documents\Sda\Code\BjaxXsl\Examples\XmlAviationWeather\jdom2-2.0.6-javadoc.jar

Download Library from Maven Repository

commons-io

Found: 19
Showing: 19

commons-io:commons-io:20030203.000550
commons-io:commons-io:2.8.0
commons-io:commons-io:2.7
commons-io:commons-io:2.6
commons-io:commons-io:2.5
commons-io:commons-io:2.4
commons-io:commons-io:2.3
commons-io:commons-io:2.2

5. [OK]
6. In Libraries > "commons.io" > [Edit ...] > Version: Release > [OK]
7. Ob

```
Maven: commons-io:commons-io:Release
```

8. Use the method in your code.

```
FilenameUtils.getBaseName(txtFileName);
```

9. Observe the imports is correctly set at the top of your module

```
import org.apache.commons.io.FilenameUtils;
```

10. Run your app to test.

Custom Library with IntelliJ IDEA

The Authorative version of "Custom Libraries" is now at AndroidJavaGradle-CustomLibrarySetup.docx

(Depcrecated) Custom Library with IntelliJ IDEA

The Authorative version of "Custom Libraries" is now at AndroidJavaGradle-CustomLibrarySetup.docx

Overview

Using Intillij IDEA our goal is to create a custom java library that we can reference from multiple java projects, while being able to edit the library code when working on a project.

The key is to add to your project the Library:

- As a module from existing sources (targeting an existing .iml file); and
- As a module dependency.

Naming and Directory Conventions

Firstly we establish our naming and directory conventions (if you don't like the following, establish your own convention).

Project, Module, and Main Entry Class name: PascalCase.

```
// Directory Stucture
..\LibraryRefDemo\MyMain

..\LibraryRefDemo\Libraries\MyJavaLibrary
..\LibraryRefDemo\Libraries\MyJavaLibrary\MyJavaLibrary.iml
..\LibraryRefDemo\Libraries\MyJavaLibrary\src\au\com\example\myjavalibrary\Main.java
```

The Main entry class shall be named "Main", rather than taking on the name of the application.

```
// The Main Entry Class base name "Main.java" is not the same as "MyJavaLibrary"
...LibraryRefDemo\Libraries\MyJavaLibrary\src\au\com\example\myjavalibrary\Main.java

package au.com.example.myjavalibrary;

public class Main {
    // The main method within the Main class ...
    public static void main(String[] args) {
        System.out.println("MyJavaLibrary! " +
au.com.example.myjavalibrary.MyStrings.GetStringFromLibrary());
    }
}
```

Package name: lowercase.

```
au.com.example.mymainproject
au.com.example.myjavalibrary
```

Gradle settings.gradle rootProject.name (or default project/module name) case:

- For a pure Java Application (e.g. not and Android app): PascalCase.

  ```
  // ..\LibraryRefDemo\MyMainApp\settings.gradle
  rootProject.name = 'MyMainApp'
  ```

- For an Android module: lowercase.

  ```
  // Generally no need for setting in a settings.gradle as case will inherit from
  // file system basename
  Project Folder PascalCase: Android\Examples\Training\MyThirdApp
  Module is lowercase:      Android\Examples\Training\MyThirdApp\app
  ```

- For a pure Java Library: lowercase.

  ```
  // ..\LibraryRefDemo\Libraries\MyJavaLibrary\settings.gradle
  rootProject.name = 'myjavalibrary'
  ```

  Pure Java Libraries may be used in an Android context. In an Android context modules are lowercase.

Gradle settings.gradle dependencies (which will generally be references to libraries) are lower case.

```
// ..\LibraryRefDemo\MyMainApp\settings.gradle
rootProject.name = 'MyMainApp'

// include line must precede projectDir setting.
// Ensure colon prefixes exist.
include ':myjavalibrary'
// It doesn't matter whether there is a trailing slash '/' or not.
project(':myjavalibrary').projectDir = new File(settingsDir, '../Libraries/MyJavaLibrary/')


// ..\LibraryRefDemo\MyMainApp\build.gradle
dependencies {
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    implementation  project(':myjavalibrary')

    // Use JUnit test framework
    testCompile 'junit:junit:4.12'
}
```

Steps (for a module dependency)

Create MyJavaLibary as an independent project and run:

1. IntelliJ > Create New Project ...
   a. Java. [Next]. [Next]
   b. Settings ...
      i. Project Name: MyJavaLibrary
      ii. Project Location: C:\Users\John\Documents\Sda\Code\Java\Examples\LibraryRefDemo\Libraries\MyJavaLibrary
   c. [Finish]
2. In the Project pane verify "Src" folder is blue, designating a "source" folder. If grey:
   a. File > Project Structure > Project Settings > Modules.
   b. Click on "MyMainApp" module > |Sources| > Right click on "src" folder. Choose [Sources]. Src folder should now be blue.
3. Add a package "au.com.example.myjavalibrary" underneath src.
4. Under "src/au.com.example.myjavalibrary" add a test class of MyJavaLibrary, that does work (return a string).

```
package au.com.example.myjavalibrary;

public class MyStrings {
    public static String GetStringFromLibrary() {
        return "Shamzam!";
    }
}
```

5. Under "src/au.com.example.myjavalibrary" add a Main class. Add code that calls the test class (MyStrings).

```
package au.com.example.myjavalibrary;

public class Main {
    public static void main(String[] args) {
        System.out.println("MyJavaLibrary! " +
au.com.example.myjavalibrary.MyStrings.GetStringFromLibrary());
    }
}
```

6. Create a run configuration. Add Cofiguration > [+] > Application:
   a. Name: jlbMyJavaLibraryRun
   b. Main Class: au.com.example.myjavalibrary.MyJavaLibrary

   c. Working Directory: C:\Users\John\Documents\Sda\Code\Java\Examples\LibraryRefDemo\Libraries\MyJavaLibrary
   d. Use classpath of module: MyJavaLibrary.
   e. [OK]
7. Run jlbMyJavaLibraryRun. Verify output:

```
MyJavaLibrary! Shamzam!
```

Create MyMainApp as an independent project and run:

1. Close the MyJavaLibrary Project.
2. Create New Project ...
   a. Java > [Next] > [Next]
   b. Project Name: MyMainApp
   c. Project Location: C:\Users\John\Documents\Sda\Code\Java**\Examples\LibraryRefDemo\MyMainApp** (Not …\LibraryRefDemo\Libraries\MyMainApp).
   d. [Finish]
3. In the Project pane verify "src" folder is blue, designating a "source" folder. If grey:
   a. File > Project Structure > Project Settings > Modules.
   b. Click on "MyMainApp" module > |Sources| > Right click on "src" folder. Choose [Sources]. Src folder should now be blue.
4. Create Package underneath "src" folder: au.com.example.mymainapp
5. Create Main class under src/au.com.example.mymainapp and add some dummy code:

```
package au.com.example.mymainapp;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

8. Create a run configuration. [+] > Application >
   a. Name: jlbMyMainAppRun
   b. Main Class: au.com.example.mymainapp.MyMainApp
   c. Working Directory: C:\Users\John\Documents\Sda\Code\Java\Examples\LibraryRefDemo\MyMainApp
   d. Use classpath of module: MyMainApp.
   e. [OK]
9. Run jlbMyMainAppRun. Verify output:

```
Hello World!
```

Add to MyMainApp Project the Library, MyJavaLibary, as a module and a module dependency:

1. Open MyMainApp as a project (which will be already open if you are continuing from above).
2. File > New ... > **Module from existing sources** {Not "Project from existing sources}...
   a. Select ..\LibraryRefDemo\Libraries\MyJavaLibrary\MyJavaLibrary.iml. [OK]
   b. Observe in the Project Pane MyJavaLibrary is added as module (and you can edit its source code).

3. (Continuing with MyMainApp open as a project) File > Project Structure … > Project Settings > Modules > MyMainApp > Dependencies > [+] > Module Dependency > Select "MyJavaLibrary". [OK]. [OK].

Verify you can reference MyJavaLibrary code from MyMainApp:

1. Change the Main class as follows

```
package au.com.example.mymainapp;

public class Main {
    public static void main(String[] args) {
                System.out.println("Hello World!");

System.out.println(au.com.example.myjavalibrary.MyStrings.GetStringFromLibrary());
    }
}
```

2. Run jlbMyMainAppRun. Observe:

```
Hello World!
Shamzam!
```

Verify you your edits to MyJavaLibrary code will be picked up by MyMainApp.

1. Edit MyJavaLibrary code. Change the test string.

```
public class MyStrings {
    public static String GetStringFromLibrary() {
        return "Shamzamxxx!";
    }
}
```

2. Run jlbMyMainAppRun. Observe:

```
Hello World!
Shamzamxxx!
```

Gradle Conversion

Assuming "Steps (for a module dependency)", above, has been followed …

Delete the native reference to MyJavaLibrary from MyMainApp. With MyMainApp open... File > Project Structure … > Project Settings > Modules > MyJavaLibrary [-]. [Yes]. [OK]. Observe MyJavaLibrary is no longer available in the Project Pane.

Change a native IntelliJ Java Library to use Gradle.

1. Close MyMainApp. Open MyJavaLibrary.
2. Create `build.gradle` in the root of MyJavaLibrary. Use `sourceSets.main.java.srcDirs = ['src']` and id 'java-library' …

```
plugins {
//    // Apply the java plugin to add support for Java
//    id 'java'
//
//    // Apply the application plugin to add support for building an application
//    id 'application'

    // Apply the java-library plugin to add support for Java Library
    id 'java-library'
}

sourceSets.main.java.srcDirs = ['src']

// Define the main class for the application
//mainClassName = 'au.com.example.myjavalibrary.Main'

dependencies {
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    // Use JUnit test framework
    testCompile 'junit:junit:4.12'
}

// In this section you declare where to find the dependencies of your project
repositories {
    // Use jcenter for resolving your dependencies.
    // You can declare any Maven/Ivy/file repository here.
    jcenter()
}
```

C:\Users\John\Documents\Sda\Code\Java\Examples\LibraryRefDemo\Libraries\MyJavaLibrary\build.gradle

Change a native IntelliJ MyMainApp to use Gradle:

1. Close MyJavaLibrary. Open MyMainApp.
2. Underneath the root directory add a `build.gradle` file with `sourceSets.main.java.srcDirs = ['src']` and `mainClassName = ''au.com.example.mymainapp.MyMainApp'.`

```
plugins {
    // Apply the java plugin to add support for Java
    id 'java'

    // Apply the application plugin to add support for building an application
    id 'application'
}

sourceSets.main.java.srcDirs = ['src']

// Define the main class for the application
mainClassName = 'au.com.example.mymainapp.Main'

dependencies {
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    // Use JUnit test framework
    testCompile 'junit:junit:4.12'
}

// In this section you declare where to find the dependencies of your project
repositories {
    // Use jcenter for resolving your dependencies.
    // You can declare any Maven/Ivy/file repository here.
    jcenter()
}
```

C:\Users\John\Documents\Sda\Code\Java\Examples\LibraryRefDemo\MyMainApp\build.gradle

3. Close the project and reopen. When offered, import Gradle changes, using defaults.
4. Allow gradle to build the project.
5. Temporarily comment out code reference to MyJavaLibrary from MyMainApp

```
package au.com.example.mymainapp;
```

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World!");
//
System.out.println(au.com.example.myjavalibrary.MyStrings.GetStringFromLibrary());
    }
}
```

6. Create a Gradle based run configuration.
    a. Gradle Pane > MyMainApp > Tasks > application > run
    b. Right click > Create 'MyMainApp [run]'. Accept Defaults. [OK]
    c. Observe in Intellij's toolbar the run configuration 'MyMainApp [run]' now exist.
7. Click on the toolbar green arrow to execute 'MyMainApp [run]'.
8. Observe expected output in the Run Pane.

```
12:56:10: Executing task 'run'...

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :run
Hello World!

BUILD SUCCESSFUL in 0s
2 actionable tasks: 2 executed
12:56:10: Task execution finished 'run'.
```

To reference MyJavaLibrary from MyMainApp:

1. Within MyMainApp's root directory create a settings.gradle file, e.g. ...\Examples\LibraryRefDemo\MyMainApp\settings.gradle, as follows:

```
rootProject.name = 'mymainapp'

// include line must precede projectDir setting.
// Ensure colon prefixes exist.
include ':myjavalibrary'
// It doesn't matter whether there is a trailing slash '/' or not.
project(':myjavalibrary').projectDir = new File(settingsDir,
'../Libraries/MyJavaLibrary/')
```

2. Within MyMainApp's build.gradle file reference MyJavaLibrary as a dependency.

```
dependencies {
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    implementation  project(':myjavalibrary')

    // Use JUnit test framework
    testCompile 'junit:junit:4.12'
}
```

C:\Users\John\Documents\Sda\Code\Java\Examples\LibraryRefDemo\MyMainApp\build.gradle

3. Close MyMainApp and reopen. When prompted import gradle changes.
4. (If not already done so) Adjust your MyMainApp code to reference MyJavaLibrary code (see above).

```
package au.com.example.mymainapp;

public class MyMainApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");

System.out.println(au.com.example.myjavalibrary.MyStrings.GetStringFromLibrary());
    }
}
```

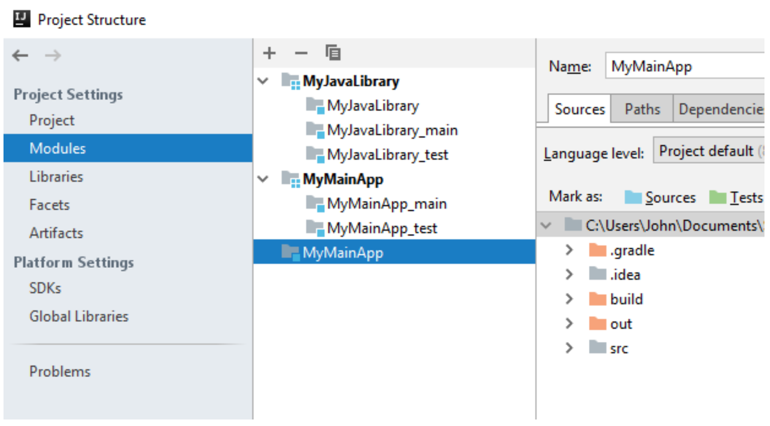
5. (If not already done so) Create a Gradle run configuration (see above).
6. Run the Gradle Configuration "MyMainApp [run]" and verify it picks up MyJavaLibrary Code.

```
12:59:01: Executing task 'run'...

> Task :myjavalibrary:compileJava UP-TO-DATE
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :myjavalibrary:processResources NO-SOURCE
> Task :myjavalibrary:classes UP-TO-DATE
> Task :myjavalibrary:jar UP-TO-DATE

> Task :run
Hello World!
Shamzamxxx!

BUILD SUCCESSFUL in 0s
4 actionable tasks: 2 executed, 2 up-to-date
12:59:01: Task execution finished 'run'.
```

7. Change the MyJavaLibrary Code, rerun MyMainApp, and verify changed MyJavaLibrary code flows through to MyMainApp.

```
Hello World!
Shamzam!
```

8. Note convoluted Intellij Native structure at File > Project Structure ... > Project Settings > Modules. Don't fuck with it.


Project Structure

Project Settings: Project, **Modules**, Libraries, Facets, Artifacts
Platform Settings: SDKs, Global Libraries
Problems

- **MyJavaLibrary**
  - MyJavaLibrary
  - MyJavaLibrary_main
  - MyJavaLibrary_test
- **MyMainApp**
  - MyMainApp_main
  - MyMainApp_test
- MyMainApp

Name: MyMainApp
Sources | Paths | Dependencie
Language level: Project default
Mark as: Sources Tests
C:\Users\John\Documents\
- .gradle
- .idea
- build
- out
- src

9. (Fix??) Add native dependency to MyJavaLibary to ensure package symbols picked up OK, as in

```
package au.com.example.mymainapp;

public class MyMainApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");


System.out.println(au.com.example.myjavalibrary.MyStrings.GetStringFromLibrary());
    }
}
```

(Deprecated) Eclipse IDE

Eclipse IDE Basics

Saving a source file (*.java) compiles it into bytecode (*.class).

- Press F11 (to debug) or Ctrl + F11 (to run).
- If you debug code and get a "source not found" error in the debug window try Resuming, F8, in order to reveal the exception.
- Another thing to try with "source not found" is to use step filtering.

Eclipse > Window > Preferences > Java > Debug > Step Filtering.
When running ensure Step Filters is set: Run > Use Step Filters (Shift + F5).

(Katz, 2012) http://www.jayway.com/2012/08/22/avoiding-source-not-found-when-i-debug-in-eclipse/

To switch back to the java perspective from the debug perspective set the following keys. This allows you to press F8 twice to terminate then switch back to the Java Perspective.

Terminate: F8, when Debugging
Show Perspective (Java): F8 when in Windows

For eclipse preference settings see "C:\Users\John\Documents\CustomData\AppDataUserSaved\Brainforest\Dev and Computer\Android Sda.pdb" > "Configure the Eclipse Environment Preferences (Window > Preferences)"

Running console applications created by eclipse

To run a console application created by eclipse ...Open your windows console in ProjectRoot\bin and type the full canonical name (packagename.filebasename) to the compiled Java program.

```
// Console application in ProjectRoot\bin
C:\Users\John\Documents\TutorialAtOracle\Regex\bin>

// Run java with the full canonical name.
C:\Users\John\Documents\TutorialAtOracle\Regex\bin>java au.com.softmake.regex.RegexTester

// Assuming
// * "RegexTester" is the name of the public class with a main method (which must be the
name of the java file);
// * "au.com.softmake.regex" is the root package name
// * \bin\au\softmake\regex\RegexTester.class is the bytecode we are attempting to run.

// Otherwise you might get
Exception in thread "main" java.lang.NoClassDefFoundError: au/com/softmake/regex/RegexTester
Caused by: java.lang.ClassNotFoundException: au.com.softmake.regex.RegexTesterBasics
```

Resource files under Eclipse compiled bytecode, that are run by Eclipse, are placed in the project root, not the bin, directory.

```
// Assuming
// C:\Users\John\Documents\Streams\ = Project Root
// C:\Users\John\Documents\Streams\bin> = Bin Directory

// Correct placement of resource file is in project root
"C:\Users\John\Documents\Streams\Xanadu.txt"

// Press F11 (Debug) in Eclipse: OK
```

Resource files under Eclipse compiled bytecode, that are run outside of eclipse, are place in the bin, not the project directory.

```
// Assuming
// C:\Users\John\Documents\Streams\ = Project Root
// C:\Users\John\Documents\Streams\bin> = Bin Directory

// Correct placement of resource file is in bin directory
"C:\Users\John\Documents\Streams\bin\Xanadu.txt"

// Run from the console
C:\Users\John\Documents\Streams\bin>java au.com.softmake.Streams

// OK
```

Basic Syntax

Comments

```
//
/* ... */
/** ... */ For use with the Javadoc tool
```

Semicolons

Semicolons ";" terminate, not separate, statements.

Main Method

Every application must have a main method:

```
public static void main(String[] args)
```

"args" are command line arguments.

Variables and Values

Types

Types of variables:

```
int gear = 1;
```

- In a class: Instance variables (non-static fields)
- In a class: Class variables (static fields).
- In a method: local variables.
- To a method: parameters.

(Oracle, 2012) http://download.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

Variables don't have to be assigned on declaration but it is good practice to.

```
// OK
int gear;

// Recommended
int gear = 1;
```

"static" turns a variable into a static variable, that is a class variable. Variables without "static" are instance variables. The "static" keyword would better have been "classvar".

```
static int wheels = 2;
```

(Oracle, 2012) http://download.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

Constants

"final" turns a variable into a constant.

```
final int numGears = 6
```

(Oracle, 2012) http://download.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

Case

Variable names are case sensitive.

DataTypes

Local variables must be assigned before used.

Else a compile-time error will result.

See "Default Values" (Oracle, 2012) http://download.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Primitive Datatypes

Datatype	Default value	Examples, Value, or Notes

byte	0	8 bit. -128 to 127 (inclusive)
short	0	16 bit. -32,768 to 32,767
int	0	32 bit. -2,147,483,648 to 2,147,483,647
long	0L	64 bit. -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	0.0f	32 bit. Don't use for currency.
double	0.0d	64 bit. Don't use for currency.
boolean	false	true/false
char	\u0000	16-bit Unicode character. '\u0000' (or 0) to '\uffff' (or 65,535 inclusive)

API supported basic datatypes

Datatype	**Default value**	**Examples, Value, or Notes**
`String`	`null`	`java.lang.String`

`BigDecimal`		`java.math.BigDecimal`
		`Use for currency`

`[An Object]`	`null`	

`Calendar`
`GregorianCalender`
`Date`
	`null`	

Date and Time operations

For Date and Time operations see:

- (Bentley, Java Reference - Framework.docx, 2013) Internationalisation, Dates and Times.
- (Bentley, Java Reference - Language - Printf Style Formatting.docx, 2013)
- (Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) DateTimesDemo.java

Basic DateTime formatting and timezone conversion.

```
public static void nowTimeZoneTranslation() {
  Calendar nowCalendar = GregorianCalendar.getInstance();
  Date nowDate = nowCalendar.getTime();

// Alternative (but Date doesn't handle complex date manipulation).
//  Date nowDate = new Date();
```

```
  SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM(MMM)-dd HH:mm:ss | Z, z, zzzz");

  TimeZone[] timezones =
      { TimeZone.getDefault(), TimeZone.getTimeZone("Australia/Perth"),
          TimeZone.getTimeZone("UTC"), TimeZone.getTimeZone("America/Chicago"),
          TimeZone.getTimeZone("Pacific/Fiji") };

  for (TimeZone timeZoneLoop : timezones) {
    sdf.setTimeZone(timeZoneLoop);
    System.out.printf("%16s | ", timeZoneLoop.getID());
    System.out.printf("%s%n", sdf.format(nowDate));
  }
}

// Output
Australia/Sydney | 2013-08(Aug)-31 19:19:04 | +1000, EST, Eastern Standard Time (New South
Wales)
 Australia/Perth | 2013-08(Aug)-31 17:19:04 | +0800, WST, Western Standard Time (Australia)
             UTC | 2013-08(Aug)-31 09:19:04 | +0000, UTC, Coordinated Universal Time
 America/Chicago | 2013-08(Aug)-31 04:19:04 | -0500, CDT, Central Daylight Time
    Pacific/Fiji | 2013-08(Aug)-31 21:19:04 | +1200, FJT, Fiji Time
```

(Bentley, Java Reference - Framework.docx, 2013) Internationalisation, Formatting Dates and Times.

Basic DateTime Parsing.

```
private static void parseDateTimes() {
  SimpleDateFormat sdfInput = new SimpleDateFormat("yyyy-MM-dd HH:mm");
  SimpleDateFormat sdfOutput = new SimpleDateFormat("EEE, dd MMM yyyy, 'at' h:mm a");
  Date dateTimeDate = null;

  try {
    dateTimeDate = sdfInput.parse("2013-06-30 21:15");
    System.out.printf("%s%n", sdfOutput.format(dateTimeDate));
  } catch (ParseException e) {
    e.printStackTrace();
  }
}

// Output
Sun, 30 Jun 2013, at 9:15 PM
```

(Bentley, Java Reference - Framework.docx, 2013) Internationalisation, Formatting Dates and Times.

The Java 6 DateTime implementation is so schizophrenic that you might find yourself needing to convert a DateTime back and forth between a Date() object and a Calendar() object.

```
private static void DateToCalendarConversions() {
  SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM(MMM)-dd HH:mm");

  Date nowDate = new Date();
  System.out.printf("%s%n", sdf.format(nowDate));

  // Convert Date to Calendar Object for any manipulation as follows.
  Calendar dateTimeCalendar = GregorianCalendar.getInstance();
  dateTimeCalendar.setTime(nowDate);
  dateTimeCalendar.set(Calendar.MONDAY, Calendar.JANUARY);

  // Output Calendar object with printf
  String printfDateTimeFormatString = "%1$tY-%1$tm(%1tb)-%1$td %1$tR%n";
  System.out.printf(printfDateTimeFormatString, dateTimeCalendar);

  // Convert Calendar back to Date Object for use with sdf.format
  Date dateTimeDate = dateTimeCalendar.getTime();
  System.out.printf("%s%n", sdf.format(dateTimeDate));
}
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) DateTimeDemo.java
(Bentley, Java Reference - Framework.docx, 2013) Internationalisation, Formatting Dates and Times.

Number, Currency and Percentage operations

For Number, Currency, and Percentage operations see:

- (Bentley, Java Reference - Language - Printf Style Formatting.docx, 2013)
- (Bentley, Java Reference - Framework.docx, 2013) Internationalisation , Formatting Numbers, Currencies and Percentages.
- (Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) Internationalisation.java

Literals

See: (Oracle, 2012) http://download.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

A *literal* is the source code representation of a fixed value.

```
byte b = 100;
short s = 10000;
int i = 100000;

boolean result = true;
char capitalC = 'C';
```

Numeric Literals

An integer literal is of type long if it ends with the letter L or l, otherwise it is of type int.

Integer literals can be expressed as decimals (base10), hexadecimals (base 16, prefix "0x"), or binaries (base 2, prefix "0b").

```
// All of the following
// represent 26.
int decimalVal = 26;
int hexadecimalVal = 0x1a;
int binanryVal = 0b11010;
```

A floating-point literal is of type float if it ends with the letter F or f; otherwise its type is double and it can optionally end with the letter D or d.

```
double d1 = 123.4;
float f1  = 123.4f;
```

The floating point types (float and double) can also be expressed using E or e (for scientific notation)

```
double d1 = 123.4;
// same value as d1, but in scientific notation
double d2 = 1.234e2;
```

Java SE 7 and above. You can use underscores, "_", between digits in numeric literals, for readability.

```
long creditCardNumber = 1234_5678_9012_3456L;
```

String literals

Literals of types char and string may contain any Unicode (UTF-16) characters. Always use 'single quotes' for char literals and "double quotes" for String literals.

```
char logoLetter = 'B';
java.lang.String bikeName = "The Gimp";
```

String literal special characters

Character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Tab
\'	Apostrophe or single quote
\"	Double quote
\\	Backslash character (\).
\uXXXX	The Unicode character specified by the four hexadecimal digits XXXX. For example, \u00A9 is the Unicode sequence for the copyright symbol.

```
java.lang.String manufacturer = "Don\"t fuck it\u00A9";
```

(Oracle, 2012) [*http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html*](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html)

Null literal

The null literal can be assigned to reference type(?), but never a primitive type.

(Oracle, 2012) [*http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html*](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html)

Class literal

A class literal, formed by taking a type name and appending ".class"; for example, String.class. This refers to the object (of type Class) that represents the type itself.

(Oracle, 2012) [*http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html*](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html)

Arrays

Zero based; Fixed length at creation; Must declare datatype.

Simple

Declare

```
int[] winnings;
```

Initialise

```
winnings = new int[10];
// '10' is the number of elements.

// Intialise with no items
String[] args = new String[0];
```

Declare and Initialise

```
int[] winnings = new int[10];
```

Assign

```
winnings[0] = 400;
winnings[1] = 350;
```

Declare, initialise and assign

```
int[] winnings = {400, 350}
```

Declare, then initialise and assign

```
String[] things; // Declare
things = new String[] {"nice", "fun"}; // Initialise and assign
```

Access

```
System.out.println(winnings[1]);
```

Get Length

```
System.out.println(anArray.length);
// For multideminsional arrays prints out the number of elements in the first dimension.
```

Array of Arrays

Declare

```
String[][] names;
```

Initialise

```
names = new String[3][2];
// '3' is the number of rows, '2' the number of columns.
```

Declare and Initialise

```
String[][] names = new String[3][2];
```

Assign

```
names[0][0] = "Joe";
names[0][1] = "Blow";
```

Declare, Initialise, and assign

```
String[][] names = {
              {"Joe", "Blow"},
              {"Lisa", "Simpson"}
};
```

Access

```
System.out.println("First Name: " + names[0][0] + " " + names[0][1]);
```

C:\Users\John\Documents\Sda\Code\Java\Examples\TutorialAtOracle\HelloWorld\src\helloworld\demo\ArrayDemo.java

Array of Objects

Array of objects example

```
private static void arrayOfObjectsDemo() {

  // A local class
  class Person {
    String givenName = "";
    String familyName = "";
    int age = 0;

    // Overrides Object.toString()
    public String toString() {
      return String.format("%s %s %d", this.givenName, this.familyName, this.age);
    }
  } // Person

  // Declare and Initialise (arrays must be fixed length)
  Person[] persons = new Person[2];

  // Each object must be created.
  persons[0] = new Person();
  persons[0].givenName = "David";
  persons[0].familyName = "Hasselhoff";
  persons[0].age = 61;

  persons[1] = new Person();
  persons[1].givenName = "Lisa";
  persons[1].familyName = "Simpson";
  persons[1].age = 12;

  for (Person person : persons) {
    System.out.println(person.toString());
  }

} // arrayOfObjectsDemo
```

Array Operations

Output

Simple array output, one line.

```
private static void simpleArrayOutputOneLine () {
  String[] fruit = {"Apples", "Oranges", "Bananas", "Pears"};
  System.out.println(Arrays.toString(fruit));
}

// Output
[Apples, Oranges, Bananas, Pears]
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) ArrayDemo.java

Simple array output, println array elements.

```
private static void printlnArrayElements(Object[] array){
  for (Object element : array) {
    System.out.printf("%s%n", element.toString());
  }
}

private static void simpleArrayOutputPrintlnArrayElements() {
  String[] fruit = { "Apples", "Oranges", "Bananas", "Pears" };
  printlnArrayElements(fruit);
}

// Output
Apples
Oranges
Bananas
Pears
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) ArrayDemo.java

Simple character array output.

```
private static void simpleCharacterArrayOutput() {
  char[] name = { 'd', 'a', 'v', 'e' };
  System.out.println(new String(name));
}

// Outputs "dave".
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) ArrayDemo.java

Copy

Copy array: use System.arrarycopy

```
public static void arraycopy(Object src, int srcPos,
                             Object dest, int destPos, int length);

private static void arrayCopyDemo() {
  char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
    'i', 'n', 'a', 't', 'e', 'd' };
  char[] copyTo = new char[7];

  System.arraycopy(copyFrom, 2, copyTo, 0, 7);
  System.out.println(new String(copyTo));
}
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) RegexDemo.java

Sort

Sort array.

```
private static void arraySortDemo() {
    char[] name = { 'd', 'a', 'v', 'e' };
    System.out.println(new String(name));
    Arrays.sort(name);
    System.out.println(new String(name));
}

// output
dave
adev
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) ArrayDemo.java
C:\Users\John\Documents\Sda\Code\Java\Examples\TutorialAtOracle\HelloWorld\src\helloworld\demo\ArrayDemo.java

Test

Test that array doesn't have any items (in that sense is empty).

```
private static void arrayTestForNoItems() {
  String[] args = new String[0];
  if (args.length != 0) {
    System.out.println("The args:");
    for (String argument: args) {
      System.out.println(argument);
    }
  }
}
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013)
C:\Users\John\Documents\Sda\Code\Java\Examples\TutorialAtOracle01\HelloWorld\src\helloworld\demo\ArrayDemo.java
See also The Platform Environment > Command line arguments

Operators

General

Arithmetic Operators (binary)	* / % + -
Arithmetic Operators (unary)	expr++ expr-- ++expr --expr +expr -expr
Boolean complement (unary)	! (inverts a boolean: not.)
Bit complement (unary)	~ (inverts a bit pattern)
Assignment Operators	= += -= *= /= %= &= ^= |= <<= >>= >>>=
Bitwise Operators	& ^ (exclusive or) | (inclusive or) << >> >>>
Comparison Operators	< > <= >= == != instanceof
Logical Operators	&& ||
Logical Operator (ternary)	? :
String Operators	+

&& as well as || are short circuit evaluated.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op2.html

The instanceof operator is a Boolean that evaluates to true if leftHandOjbect an instance of a: class; inherited class; or interface.

```
class InstanceofDemo {
    public static void main(String[] args) {

        Parent obj1 = new Parent();
        Parent obj2 = new Child();
        System.out.println("obj2 instanceof Parent: "
            + (obj2 instanceof Parent));
        System.out.println("obj2 instanceof Child: "
            + (obj2 instanceof Child));
        System.out.println("obj2 instanceof MyInterface: "
            + (obj2 instanceof MyInterface));
    }
}

class Parent {}
class Child extends Parent implements MyInterface {}
interface MyInterface {}

obj2 instanceof Parent: true
obj2 instanceof Child: true
obj2 instanceof MyInterface: true
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op2.html

Bit Flag Operation Replacment

EnumSets afford a typesafe replacement for bit flag (bitflag) operations. See Java%20Reference%20-%20Framework.docx#EnumSetBitFlagReplacement

Statements, Blocks and Expressions

Overview

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html

Statements are a complete unit of execution.

There are three types of statements: expression statements; declaration statements; and control flow statements.

```
// Expression statements
apples += 12;
apples = 5;

// Declaration statements
double height = 5.1;

// Control flow statements
if (x < 10) then {
  // do something
}
```

A block is a group of zero or more statements between braces.

Expressions evaluate to a single value.

Expression evaluate according to an order of precedence. Use parenthesis, (and), to force an order of evaluation.

Control Flow Statements

Control flow statements include: decision making statements (if-then, if-then-else, switch); looping statements (for, while, do-while); and branching statements (break, continue, return)

If

if-then. Conventional form has braces (recommended).

```
// Conventional form
if (isMoving) {
      currentSpeed--;
}
```

if-then. Short form omits braces (discouraged).

```
// Short form over two lines (discouraged)
if (isMoving)
  currentSpeed--;
```

```
// Short form over one line (discouraged)
if (isMoving) currentSpeed--;
```

if-then-else. "else if" not run together (i.e. not "elseif").

```
if (testscore >= 90) {
        grade = 'A';
  } else if (testscore >= 80) {
        grade = 'B';
  } else {
        grade = 'F';
}
```

(Oracle, 2012) [*http://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html*](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html)

Switch

Table 1 Switch works with the following data types

Data Type Group	**Data Type**
Primative data types	byte, short, char, int
Special Classes that wrap primatives	Byte, Short, Character, Integer
Enumerated Types	
String Class (> SE7)	

(Oracle, 2012) [*http://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html*](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html)

Switch example

```
int month = 2;
String monthString = "";
switch (month) {
      case 1:
             monthString = "Jan";
             break;
      case 2:
             monthString = "Feb";
             break;
      default:
             monthString = "Invalid Month";
             break;
} // switch
```

The switch statement evaluates its expression, then executes all statements that follow the matching case label. Each break statement terminates the enclosing switch statement. Without break all the statements from the first case that matches will be evaluated

```
int month = 8;
switch (month) {
  case 1:  futureMonths.add("January");
  case 2:  futureMonths.add("February");
  case 3:  futureMonths.add("March");
  case 4:  futureMonths.add("April");
  case 5:  futureMonths.add("May");
  case 6:  futureMonths.add("June");
  case 7:  futureMonths.add("July");
  case 8:  futureMonths.add("August");
  case 9:  futureMonths.add("September");
  case 10: futureMonths.add("October");
  case 11: futureMonths.add("November");
  case 12: futureMonths.add("December");
```

```
            break;
  default: break;
}

// This is the output from the code:
August
September
October
November
December
```

Use switch with multiple conditions for a particular result

```
switch (month) {
      case 1: case 3: case 5:
      case 7: case 8: case 10:
      case 12:
          numDays = 31;
          break;
      case 4: case 6:
      case 9: case 11:
          numDays = 30;
          break;
      case 2:
              System.out.println("Feb not yet implemented.");
          break;
      default:
          System.out.println("Invalid month.");
          break;
}
```

From Jave SE7 you can use a String object in a switch statement.

```
switch (month.toLowerCase()) {
case "january":
      monthNumber = 1;
      break;
case "february":
      monthNumber = 2;
      break;
case "march":
      monthNumber = 3;
      break;
default:
      monthNumber = 0;
      break;
}
```

Iteration

While and do-while

Basic syntax

```
while (expression) {
     // statement(s)
}

do {
     statement(s)
} while (expression);
```

For

Basic syntax

Basic syntax eample

```
for (initialization; termination; increment) {
    statement(s)
}

for (int i = 1; i < 11; i++) {
      System.out.println("Count is: " + i);
}
```

The scope of the initialisation variable extends from its declaration to the end of the block governed by the for statement (so if you need to reference it outside the loop, declare it outside the loop).

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

Multiple declarations can be comma separated

```
for (int i = 0, n = elements.length; i < n; i++) {
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/chained.html

Enhanced-for Arrays

The "enhanced-for" allows you to iterate over arrays, enums, and collections.

Enhanced for iteration: arrays.

```
// Iterate over and array of primitive datatypes
int[] numbers = {1,2,3,4,5,6};
java.lang.String str = "";
for (int item : numbers) {
      str = str + item + ", ";
}
```

Enhanced-for iteration: array of objects.

```
// Declare and Initialise
Person[] persons = new Person[2];

// Each object must be created.
persons[0] = new Person();

// Object assignment
persons[0].givenName = "David";
persons[0].familyName = "Hasselhoff";
persons[0].age = 61;

persons[1] = new Person();
persons[1].givenName = "Lisa";
persons[1].familyName = "Simpson";
persons[1].age = 12;

for (Person person : persons) {
  System.out.println(person.toString());
}
```

Enhanced-for Enums

Enhanced-for iteration: enums

```
private enum Day {
    SUNDAY,
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY
}

static private void EnumSetImplementation() {
    for (Day d : Day.values()){
        System.out.println(d);
    }
}

// Output
SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/collections/implementations/set.html C:\Users\John\Documents\Sda\Code\Java\Examples\TutorialAtOracle\HelloWorld\src\helloworld\demo\collections\ImplementationsDemo.java

Enhanced-for iterations: iterate over enum range with EnumSet Collection implementation. See Java%20Reference%20-%20Framework.docx#EnumSetRange

(Bentley, Java Reference - Framework.docx, 2013)

Enhanced-For Collections

Enhanced-for iteration: Collections. Iterate a collection in four ways:

aggregate operations (from Java 8);

Enhanced-for, elements referenced by object;

```
for (Object obj : sortedSet) {
    System.out.println(obj.toString());
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/collections/interfaces/collection.html

Enhanced-for, elements referenced by type;

```
for (String s : sortedSet) {
    System.out.println(s);
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/collections/interfaces/collection.html

Enhanced-for, iterators.

```
static void filter(Collection<?> c) {
    for (Iterator<?> it = c.iterator(); it.hasNext(); )
        if (!cond(it.next()))
            it.remove();
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/collections/interfaces/collection.html

Enumeration object (deprecated "New implementations should consider using Iterator in preference to Enumeration.")

```
tags = new Hashtable;

// .... Insert keys and values ...

Enumeration enumeration = tags.keys();

String tagName = "";
int count = 0;
while (enumeration.hasMoreElements()) {
    tagName = (String) enumeration.nextElement();
    count = ((Integer) tags.get(tagName)).intValue();
    System.out.format("Local Name '%s' occurs %d times.%n", tagName, count);
}
```

(Oracle, 2011) http://docs.oracle.com/javase/7/docs/api/java/util/Enumeration.html
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html

See Java%20Reference%20-%20Framework.docx#CollectionIteration

Branching Statements

Break

break statements can operate on switch, for, while, or do-while.

break statements are unlabeled or labeled.

An unlabeled break terminates execution of the innermost statement and returns execution after the loop it terminates

```
for (i = 0; i < arrayOfInts.length; i++) {
  if (arrayOfInts[i] == searchfor) {
    foundIt = true;
    break;
  }
}
// Execution resumes here.
```

A labeled break terminates execution of the loop with has been labeled and returns execution after this statement.

```
search:
  for (i = 0; i < arrayOfInts.length; i++) {
    for (j = 0; j < arrayOfInts[i].length; j++) {
      if (arrayOfInts[i][j] == searchfor) {
        foundIt = true;
        break search;
      }
    }
  }
  // Execution resumes here.
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html

Continue

The continue statement skips the current iteration of a for, while , or do-while loop.

The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop.

A labeled continue statement skips the current iteration of an outer loop marked with the given label.

Return

The return statement exits from the current method and control flow returns to where the method was invoked.

```
private static void returnDemo() {
  int x = 2;
  System.out.println("Dog");

  if (x == 2) {
    return;
    // Execution jumps out of method.
  }
  System.out.println("Cat");
}
```

The return statement has two forms: one that returns a value, and one that doesn't. If it returns a value it's datatype must match that declared by the method.

Objects And Classes

Intro

Object and class basics

Class Declare

```
package helloworld;

public class Bicycle {
    public String manufacturer = "";
    public String modelName = "";
    public int cadence = 0;
    public int speed = 0;
    public final int numberOfGears = 12;

    public Bicycle() {
        this.cadence = 1;
        this.speed = 22;
    }
    public Bicycle(String manufacturer, String modelName, int cadence, int speed) {
        this.manufacturer = manufacturer;
        this.modelName = modelName;
        this.cadence = cadence;
        this.speed = speed;
    }

    public void changeCadence(int newValue) {
        cadence = newValue;
    }

    public void speedUp(int increment) {
        speed = speed + increment;
    }

    @Override
    public String toString() {
        return String.format("(%s %s) cadence: %d speed: %d number of gears: %d %n",
                             manufacturer,
                             modelName,
                             cadence,
                             speed,
                             numberOfGears);
    }
}
```

C:\Users\John\Documents\Sda\Code\Java\Examples\TutorialAtOracle\HelloWorld\src\helloworld\Bicycle.java

Object Creation

```
Bicycle bike1 = new Bicycle();
```

Method Invocation

```
bike1.changeCadence(50);
bike1.speedUp(10);
bike1.printStates();
```

Property (or "Field") Invocation

```
System.out.println("speed " + bike1.speed);
```

Inheritance

```
class MountainBike extends Bicycle {

     // new fields and methods defining a mountain bike would go here

}
```

Interface basics

Declare an interface

```
public interface IBicycle {
  void changeGear(int newValue);
  void speedUp(int increment);
}
```

Implement an interface

```
public class MyCoolBicycle implements IBicycle {
      int speed = 0;
      int gear = 0;

  public void changeGear(int newValue) {
      gear = newValue;
  }

  public void speedUp(int increment){
      speed = speed + increment;
  }

  public String getStates(){
      return "gear: " + gear + " speed: " + speed;
  }
}
```

Use class that has implemented an interface

```
MyCoolBicycle bike1 = new MyCoolBicycle();

bike1.changeGear(50);
bike1.speedUp(10);

System.out.println(bike1.getStates());
```

Overloading Methods

Two of the components of a method declaration comprise the *method signature*—the method's name and the parameter types.

```
calculateAnswer(double, int, double, double)
```

You can overload methods (of the same name) by having different method signatures

```
public class DataArtist {
    ...
    public void draw(String s) {
        ...
    }
    public void draw(int i) {
        ...
    }
    public void draw(double f) {
        ...
    }
    public void draw(int i, double f) {
        ...
    }
}
```

Method Chaining

Method chaining is where you first invoke a method that returns an object. You then immediately invoke a method on that object, which returns yet another object, and so on.

```
String value = Charset.defaultCharset().decode(buf).toString();
```

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/essential/io/fileOps.html#chaining*

You can also do multiline method chaining.

```
    // Pass null as the parent view because its going in the dialog layout
    builder.setView(inflater.inflate(R.layout.dialog_signin, null))
    // Add action buttons
           .setPositiveButton(......
```

http://developer.android.com/guide/topics/ui/dialogs.html#AddingAList

To support method chaining by a method in your class: rather than returning nothing; return this;

```
// Doesn't support method chaining
public void makeText(String text) {
    this.text = text;
}

// Supports method chaining
public Toast makeText(String text) {
    this.text = text;
    return this;
}
```

http://stackoverflow.com/questions/2872222/how-to-do-method-chaining-in-java-o-m1-m2-m3-m4 > *Answer by Thomas Lötzer at 2010-05-20 08:55*

Pros and Cons of method chaining:

Pro

- "This technique produces compact code and enables you to avoid declaring temporary variables that you don't need."

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/essential/io/fileOps.html#chaining*

Constructors

Constructors take the name of the class and have no return type.

```
public class Bicycle {
  int cadence = 0;
  int speed = 0;

  public Bicycle(){
    cadence = 1;
    speed = 1;
  }
```

Constructors can be overloaded (so you can have two constructors so long as their signatures are different).

All classes have at least one constructor. If a class does not explicitly declare any, the Java compiler automatically provides a no-argument constructor, called the default constructor. This default constructor calls the class parent's no-argument constructor, or the Object constructor if the class has no other parent. If the parent has no constructor (Object does have one), the compiler will reject the program.

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/java/javaOO/objectcreation.html*

Constructors have access modifiers.

Parameters and Arguments

Parameters refers to the list of variables in a method declaration. *Arguments* are the actual values that are passed in when the method is invoked.

Variable arguments are achieved with the *varargs* construction. To use follow the datatype by ellipses (…). Within the method the *vararg* is treated as an array. You can pass arguments to the *vararg* parameter as an array or as a comma separated list.

```
public Polygon polygonFrom(Point... corners) {
    int numberOfSides = corners.length;
    double squareOfSide1, lengthOfSide1;
      squareOfSide1 = (corners[1].x - corners[0].x)
                     * (corners[1].x - corners[0].x)
                     + (corners[1].y - corners[0].y)
                     * (corners[1].y - corners[0].y);
    lengthOfSide1 = Math.sqrt(squareOfSide1);
```

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/java/javaOO/arguments.html*

A parameter can have the same name as one of the class's fields. If this is the case, the parameter is said to *shadow* the field. Shadowing fields can make your code difficult to read and is conventionally used only within constructors and methods that set a particular field.

```
public class Circle {
    private int x, y, radius;
    public void setOrigin(int x, int y) {
        ...
    }
}
```

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/java/javaOO/arguments.html*

Each method parameter shadows the field that shares its name. So using the simple names `x` or `y` within the body of the method refers to the parameter, *not* to the field. To access the field, you must use a qualified name.

Primitive arguments, such as an int or a double, are passed into methods by value.

> This means that any changes to the values of the parameters exist only within the scope of the method. When the method returns, the parameters are gone and any changes to them are lost.

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/java/javaOO/arguments.html*

Reference data type parameters, such as objects, are also passed into methods *by value*. This means that when the method returns, the passed-in reference still references the same object as before. *However*, the values of the object's fields *can* be changed in the method, if they have the proper access level.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/arguments.html

Parameter default values. Java doesn't support parameter default values as such. But the best way to simulate this is with overloading.

```
    public static void start()  {
        OutputClassMembers("java.nio.channels.ReadableByteChannel");
    }

    private enum ClassMember { CONSTRUCTOR, FIELD, METHOD, CLASS, ALL }

    private static void OutputClassMembers(String className) {
        OutputClassMembers(className, ClassMember.ALL);
    }

    private static void OutputClassMembers(String className, ClassMember classMember ) {
        try { ...
```

(Stackoverflow, 2018) , "Does Java support default parameter values?", https://stackoverflow.com/questions/997482/does-java-support-default-parameter-values
C:\Users\John\Documents\Sda\Code\Java\Examples\TutorialAtOracle\HelloWorld\src\helloworld\demo\reflection\GetClassMembers.java

Creating objects

Creating an object generally entails: Declaration (specify variable name and type), Instantiation (create an object instance); and Initialization (setting initial values).

Objects can be instantiated without being assigned to a variable.

```
Collator fr_FRCollator = Collator.getInstance(new Locale("fr","FR"));

// The object instance field is assigned to a variable, not the object instance.
int height = new Rectangle().height;

// or
System.out.println(new Rectangle(100, 50).getArea());
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/objectcreation.html

Destruction of Objects

You don't have to explicitly destroy an object. Java automatically destroys and object through *garbage collection.* An object is eligible for garbage collection when it has no (variable) references that point to it. Objects references are dropped when the object variable goes out of scope or you set the object variable to null.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/usingobject.html

Returning

Methods declared with void don't return a value but you can use return; to branch out of statements and terminate the method.

You can return a class or an interface.

When a method uses a class name as its return type the class of the type of the returned object must be either a subclass of, or the exact class of, the return type.

```
public Number returnANumber() {
    ...
}
```

The returnANumber method can return an ImaginaryNumber but not an Object. ImaginaryNumber is a Number because it's a subclass of Number

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/returnvalue.html

You can override a method and define it to return a subclass of the original method, like this

```
public ImaginaryNumber returnANumber() {
    ...
}
```

This technique, called *covariant return type*

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/returnvalue.html

The this keyword

Within a method or a constructor this is a reference to the current object. You can refer to any member (any method, constructor or field) of the current object from within a method or a constructor by using this.

```
public class Point {
    public int x = 0;
    public int y = 0;

    //constructor
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html

From within a constructor, you can also use the this keyword to call another constructor in the same class. Doing so is called an *explicit constructor invocation*. If present the invocation of another constructor must be the first line in the constructor.

```
public class Rectangle {
    private int x, y;
    private int width, height;

    public Rectangle() {
        this(0, 0, 0, 0);
    }
    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }
    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
    ...
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html

Class modifiers

Overview

A class can have the following modifiers:

Modifier	Meaning
`public, protected, private`	Access modifier
`abstract`	The class requires an override
`static`	Restrict class to one instance (?? enums)
`final`	prevent value modification
`strictfp`	Force strict floating point behaviour
	Annotations

"Not all modifiers are allowed on all classes, for example an interface cannot be final and an enum cannot be abstract."

(Oracle, 2012), "Examining Class Modifiers and Types", https://docs.oracle.com/javase/tutorial/reflect/class/classModifiers.html

Access Modifiers

There are two levels of access control:

- At the top (class) level—`public`, or package-private (no explicit modifier).
- At the member (fields and methods) level—`public`, `private`, package-private (no explicit modifier), or `protected`.

A nested class, being a member, can be declared public, private, protected, or package private. (Note that outer classes can only be declared public or package private.)

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

For members (fields and methods) the `private` access modifier means the member can only be accessed in its own class.

For members (fields and methods) the `protected` access modifier means the member can only be accessed within its own package, as with *package-private,* and, in addition to *package-private,* by a subclass of its class in another package.

Access of class members from other classes (or own class) summary.

Modifier	Own Class	Package	Subclass (outside package)	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier (package private)	Y	Y	N	N

private	Y	N	N	N

The first data column indicates whether the class itself has access to the member defined by the access level. As you can see, a class always has access to its own members. The second column indicates whether classes in the same package as the class (regardless of their parentage) have access to the member. The third column indicates whether subclasses of the class declared outside this package have access to the member. The fourth column indicates whether all classes have access to the member.

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html*

Best practice for access modifiers:

- Use the most restrictive access level that makes sense for a particular member.
- Avoid public fields except for constants

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html*

Instance v class members

A "class" or "static" member are fields or methods associated with the class rather an instance object created from the class.

A class variable (field) can have its value changed and can be done so without an instance of a class having to be created.

Class variables are referenced using the class name rather than an instance variable of the object.

```
// Recommended class variable reference
Bicycle.numberOfBicycles

// Legal, but discouraged, class variable reference
myBike.numberOfBicycles
```

Static methods are referenced using the class name rather than an instance variable of the object.

```
// Recommended class variable reference
ClassName.methodName(args)

// Legal, but discouraged, class variable reference
instanceName.methodName(args)
```

A common use for class (static) methods is to access static fields.

Class (static) methods ***cannot*** access instance variables or instance methods directly—they must use an object reference.

Static constants are designated with the final modifier and the static modifier

```
static final double PI = 3.141592653589793;
```

JLB: But recall without the static modifier we have an instance constant. E.g. "final double PI = 3.141592653589793;" compiles ok and is an instance constant.

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html*

Initialising Fields

You can initialise fields:

- Outside a method, for simple initialisation; or
- Within a method, for complex initialisation (e.g. entailing logic or error handling).

For class (static) variables you can initialise in a method using either a "static initialisation block" or a private static method.

```
// Static intialisation block
static {
    // whatever code is needed for initialization goes here
}

// Static method
class Whatever {
    public static varType myVar = initializeClassVariable();

    private static varType initializeClassVariable() {

        // initialization code goes here
    }
}
```

For instance variables you can initialize by the following methods: simply, outside a class; in a constructor; initializer blocks; or in a final method

```
// Simple, outside a class
public static int capacity = 10;

// In a constructor
public class Point {
    public int x = 0;
    public int y = 0;

    //constructor
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

// An initializer block
{
    // whatever code is needed for initialization goes here
}

// A final method
class Whatever {
    private varType myVar = initializeInstanceVariable();

    protected final varType initializeInstanceVariable() {

        // initialization code goes here
    }
}
```

A *final method* cannot be overridden in a subclass.

Nested Classes

General

Nested classes are divided into two categories: *static* and *non-static*. Nested classes that are declared static are simply called "*static nested classes*". Non-static nested classes are called "*inner classes*".

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }

    // A non-static nested class. Aka "inner class".
    class InnerClass {
        ...
    }
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

A nested class is a member of its enclosing class.

Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private. Static nested classes do not have access to other members of the enclosing class.

A nested class can be declared private, public, protected, or package private. (Recall that outer classes can only be declared public or package private.)

Accessing static nested classes

```
OuterClass.StaticNestedClass

// For example, to create an object for
// the static nested class, use this syntax:
OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();
```

An inner class is associated with an instance of its enclosing class and has direct access to that object's methods and fields. Also, because an inner class is associated with an instance, it cannot define any static members itself.

To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object with this syntax:

```
OuterClass outerObject = new OuterClass();
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```

Nested class shadowing: If a declaration of a type (such as a member variable or a parameter name) in a particular scope (such as an inner class or a method definition) has the same name as another declaration in the enclosing scope, then the declaration shadows the declaration of the enclosing scope. You must use explicit references to get the right declaration value.

```
public class ShadowTest {

    public int x = 0;

    class FirstLevel {
        public int x = 1;

        void methodInFirstLevel(int x) {
            System.out.println("x = " + x);
            System.out.println("this.x = " + this.x);
            System.out.println("ShadowTest.this.x = " + ShadowTest.this.x);
        }
    }

    public static void main(String... args) {
        ShadowTest st = new ShadowTest();
        ShadowTest.FirstLevel fl = st.new FirstLevel();
        fl.methodInFirstLevel(23);
    }
}

// Output
x = 23
this.x = 1
ShadowTest.this.x = 0
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

There are two types of special inner classes: local inner classes (declared within the body of a method); and anonymous inner classes (declared within the body of a method without being named).

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

Local classes

Local classes are classes that are defined in a block, which is a group of zero or more statements between balanced braces. For example, you can define a local class in a method body, a `for` loop, or an `if` clause. You typically find local classes defined in the body of a method.

```
public class LocalClassDemo {

  public static void startMethod() {
    final int startMethodConstant = 5;

    class LocalClass {
      int localClassVariable;

      // LocalClass constructor
      LocalClass() {
        localClassVariable = 10;
      }

      public int get() {
        return localClassVariable + startMethodConstant;
      }
    } // LocalClass

    LocalClass lc = new LocalClass();
    System.out.println(lc.get());
  } // startMethod()

} // LocalClassDemo

// Output
15
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/localclasses.html

A local class can access members of its enclosing class.

A local class can access constants, variables declared with final, of the enclosing method.

Local classes are non-static because they have access to instance members of the enclosing block.

Local classes are like inner classes as they cannot define or declare any static members, except for constants.

```
// OK
public void sayGoodbyeInEnglish() {
  class EnglishGoodbye {
      public static final String farewell = "Bye bye";
      public void sayGoodbye() {
          System.out.println(farewell);
      }
  }
  EnglishGoodbye myEnglishGoodbye = new EnglishGoodbye();
  myEnglishGoodbye.sayGoodbye();
}
```

You can't declare an interface in a block (so no Local Interfaces).

ToDo: Java SE 8 features.

(Oracle, 2012) [*http://docs.oracle.com/javase/tutorial/java/javaOO/localclasses.html*](http://docs.oracle.com/javase/tutorial/java/javaOO/localclasses.html)

Anonymous Classes

An anonymous class is: a nameless class; declared and instantiated at once; part of an expression (rather than a class declaration); and implements an interface or extends a class.

```
public class HelloWorldAnonymousClasses {

  interface HelloWorld {
    public void greet();

    public void greetSomeone(String someone);
  }

  public void sayHello() {

    // Regular local class implementing an interface.
    class EnglishGreeting implements HelloWorld {
      String name = "world";

      public void greet() {
        greetSomeone("world");
      }

      public void greetSomeone(String someone) {
        name = someone;
        System.out.println("Hello " + name);
      }
    }

    HelloWorld englishGreeting = new EnglishGreeting();

    // Anonymous class implementing an interface.
    HelloWorld frenchGreeting = new HelloWorld() {
      String name = "tout le monde";

      public void greet() {
        greetSomeone("tout le monde");
      }

      public void greetSomeone(String someone) {
        name = someone;
        System.out.println("Salut " + name);
      }
    };

    englishGreeting.greet();
    frenchGreeting.greetSomeone("Fred");

  }

  public static void start() {
    HelloWorldAnonymousClasses helloWorldAnonymousClasses = new
HelloWorldAnonymousClasses();
    helloWorldAnonymousClasses.sayHello();
  }
}
```

Anonymous class syntax entails: the new operator; the name of the interface to implement or a class to extend; parenthesis with arguments to the constructor, if any, (when implementing interfaces you use an empty pair of parenthesis); class member declaration.

Note an anonymous class must either implement an interface or extend an existing class.

You cannot declare constructors in an anonymous class.

You can assign an anonymous class: to a variable; passed as an argument; or to a variable then passed as an argument.

```
// Anonymous class assigned to a variable
HelloWorld frenchGreeting = new HelloWorld() {
  String name = "tout le monde";

  public void greet() {
    greetSomeone("tout le monde");
  }

  public void greetSomeone(String someone) {
    name = someone;
    System.out.println("Salut " + name);
  }
};


// Anonymous class passed as an argument.
btn.setOnAction(new EventHandler<ActionEvent>() {

  @Override
  public void handle(ActionEvent event) {
      System.out.println("Hello World!");
  }
});


// Anonymous class assigned to a variable then passed as an argument.
EventHandler<ActionEvent> eventHandler = new EventHandler<ActionEvent>() {

  @Override
  public void handle(ActionEvent event) {
      System.out.println("Hello World!");
  }
};

btn.setOnAction(eventHandler);
```

Because an anonymous class definition is an expression, it must be part of a statement (rather than a standalone class declaration). That's why in frenchGreeting example there is a terminating semi-colon after the closing bracket. Note parameter expresssions, as in the setOnAction example, do not have terminating semi colon.

Enum Types

An enum type is a type whose fields consist of a fixed set of constants

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

The enum declaration defines a class (called an enum type). The enum class body can include methods and other fields.

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7);

    private final double mass;   // in kilograms
    private final double radius; // in meters
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
    private double mass() { return mass; }
    private double radius() { return radius; }

    // universal gravitational constant  (m3 kg-1 s-2)
    public static final double G = 6.67300E-11;

    double surfaceGravity() {
        return G * mass / (radius * radius);
    }
```

…

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

The compiler automatically adds some special methods when it creates an enum:

```
// The static values method contains an array of all the values that can be used as follows
for (Planet p : Planet.values()) {
    System.out.printf("Your weight on %s is %f%n",p, p.surfaceWeight(mass));
}
```

For enum iteration see Enhanced-for Enums.

Annotations

Using annotations

Annotations are applied to various programming elements (classes, fields, methods, and more).

The annotation appears first, often (by convention) on its own line, and may include elements with named or unnamed values.

```
// Elements with name/value pairs
@Author(
   name = "Benjamin Franklin",
   date = "3/27/2003"
)
class MyClass() { }

// An element with one name/value pair
@SuppressWarnings(value = "unchecked")
void myMethod() { }

// A value only
@SuppressWarnings("unchecked")
void myMethod() { }

// With no elements
@Override
void mySuperMethod() { }
```

Creating Annotations

Define the *annotation type* with an "@", keyword `interface`, and *annotation type elements* with optional default values; like this:

```
@interface ClassPreamble {
   String author();
   String date();
   int currentRevision() default 1;
   String lastModified() default "N/A";
   String lastModifiedBy() default "N/A";
   // Note use of array
   String[] reviewers();
}

// Used like this
@ClassPreamble (
   author = "John Doe",
   date = "3/17/2002",
   currentRevision = 6,
   lastModified = "4/12/2004",
   lastModifiedBy = "Jane Doe",
   // Note array notation
   reviewers = {"Alice", "Bob", "Cindy"}
)
public class Generation3List extends Generation2List {
// class code goes here
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/annotations.html

Annotations Defined by Java

@Deprecated marks the element as deprecated.

```
/**
 * @deprecated
 * An explanation of why it was deprecated
 */
@Deprecated
static void deprecatedMethod() { }
```

@Override marks a method in a subclass as one that overrides a method in the superclass.

```
@Override
int overridingMethod() { }
```

@SuppressWarnings suppressing warnings by the compiler

```
@SuppressWarnings("deprecation")
void useDeprecatedMethod() {
    // deprecation warning
    // - suppressed
   objectOne.deprecatedMethod();
}

// Another example
@SuppressWarnings({"unchecked", "deprecation"})
```

Annotation Processing

From Java 6 annotations can be processed at runtime. To do this, mark the annotation with an annotation itself @Retention(RetentionPolicy.RUNTIME):

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@interface AnnotationForRuntime {

   // Elements that give information
   // for runtime processing

}
```

Inheritance and Interfaces

Inherit and implement interface:

```
class MyClass extends MySuperClass implements YourInterface {
    // field, constructor, and
    // method declarations
}
```

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/java/javaOO/classdecl.html*

A class can *implement* more than one interface.

Interfaces Detail

An interface can contain only constants, method signatures, and nested types. There are no method bodies. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces.

The concept of interfaces:

There are a number of situations in software engineering when it is important for disparate groups of programmers to agree to a "contract" that spells out how their software interacts. *interfaces* are such contracts.

In the robotic car example above, it is the automobile manufacturers who will implement the interface. Chevrolet's implementation will be substantially different from that of Toyota, of course, but both manufacturers will adhere to the same interface. The guidance manufacturers, who are the clients of the interface, will build systems that use GPS data on a car's location, digital street maps, and traffic data to drive the car. In so doing, the guidance systems will invoke the interface methods: turn, change lanes, brake, accelerate, and so forth.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html

Once published never change an interface, instead extend an interface. Extending an interface:

```
public interface GroupedInterface extends Interface1, Interface2, Interface3 {

    // constant declarations

    // base of natural logarithms
    double E = 2.718282;

    // method signatures
    void doSomething (int i, double x);
    int doSomethingElse(String s);
}
```

http://docs.oracle.com/javase/tutorial/java/IandI/interfaceDef.html

All methods declared in an interface are implicitly public, so the public modifier can be omitted (jlb: but include it for readability).

All constant values defined in an interface are implicitly public, static, and final. These modifiers can be omitted (jlb: but include them for readability).

In Java, a class can inherit from only one class but it can implement more than one interface. Therefore, objects can have multiple types: the type of their own class and the types of all the interfaces that they implement.

```
public class MultipleInterfaces implements InterFaceOne, InterFaceTwo {

    private InterFaceOne if1;
    private InterFaceTwo if2;

    public MultipleInterfaces() {
      if1 = new ImplementingClassOne();
      if2 = new ImplementingClassTwo();
    }

    @Override
    public void classOneMethodOne { if1.methodOne(); }
    @Override
    public void classOneMethodTwo { if1.methodTwo(); }
    /** Etc. */


    @Override
    public void classTwoMethodOne { if2.methodOne(); }
    @Override
    public void classTwoMethodTwo { if2.methodTwo(); }
    /** Etc. */

}
```

StackOverflow > 2010-12-28 > Chuck Mosher > http://stackoverflow.com/questions/4546807/implementing-multiple-interfaces-with-java-is-there-a-way-to-delegate

You can use interface names anywhere you can use any other data type name. If you define a reference variable whose type is an interface, any object you assign to it *must* be an instance of a class that implements the interface.

Inheritance

Sub and Super classes

Subclasses (inherited classes) inherit all the fields and methods of the parent class (called a "superclass" in java).

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/javaOO/classes.html

In the absence of any other explicit superclass, every class is implicitly a subclass of Object.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Constructors and Inheritance

Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

```
public class MountainBike extends Bicycle {

    // the MountainBike subclass adds
    // one field
    public int seatHeight;

    // the MountainBike subclass has one
    // constructor
    public MountainBike(int startHeight,
                        int startCadence,
                        int startSpeed,
                        int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }
}
```

Overriding and Hiding methods

You can write a new instance method in the subclass that has the same signature as the one in the superclass, thus *overriding* it.

You can write a new static method in the subclass that has the same signature as the one in the superclass, thus *hiding* it.

The distinction between hiding and overriding has important implications. The version of the overridden method that gets invoked is the one in the subclass. The version of the hidden method that gets invoked depends on whether it is invoked using a reference to the superclass or the subclass.

```
Cat myCat = new Cat();
Animal myAnimal = myCat;

// References the superclass hidden class method.
Animal.testClassMethod();

// References the subclass instance overriding method
```

```
myAnimal.testInstanceMethod();

// The output from this program is as follows:

// The class method in Animal.
// The instance method in Cat.
```

You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword super.

Implicit casting

```
// obj becomes a MountainBike
Object obj = new MountainBike();
```

Explicit casting

```
MountainBike myBike = (MountainBike)obj;
```

Prevent runtime error due to improper casting

```
if (obj instanceof MountainBike) {
    MountainBike myBike = (MountainBike)obj;
}
```

An overriding method can also return a subtype of the type returned by the overridden method. This is called a *covariant return type*.

The access specifier for an overriding method can allow more, but not less, access than the overridden method. For example, a protected instance method in the superclass can be made public, but not private, in the subclass.

In a subclass, you can overload the methods inherited from the superclass. Do this with different method signatures.

Polymorphism

Polymorphism is the ability of an object to have many forms. In the object oriented (and java) context it is the ability of an instance variable of a superclass to reference the right subclass overriding methods:

```
Bicycle bike01, bike02, bike03;

bike01 = new Bicycle(20, 10, 1);
bike02 = new MountainBike(20, 10, 5, "Dual");
bike03 = new RoadBike(40, 20, 8, 23);

// Uses Bicycle.printDescription()
bike01.printDescription();

// Uses MountainBike.printDescription()
bike02.printDescription();

// Uses RoadBike.printDescription()
bike03.printDescription();
```

The following is the output from the test program:

```
Bike is in gear 1 with a cadence of 20 and travelling at a speed of 10.
Bike is in gear 5 with a cadence of 20 and travelling at a speed of 10.
The MountainBike has a Dual suspension.
```

Bike is in gear 8 with a cadence of 40 and travelling at a speed of 20.
The RoadBike has 23 MM tires.

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html*

You can, but you shouldn't, hide fields.

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/java/IandI/hidevariables.html*

Using the Keyword super

Reference an overridden superclass method with super

```
super.printMethod();
```

Call superclass constructor like this

```
super();

// Or
super(parameter list);
```

If a constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass. If the super class does not have a no-argument constructor, you will get a compile-time error.

Object class, the topmost Superclass

The object class, at the top of the object hierarchy provides the following methods that you may want to override:

```
clone()
equals()
finalize
getClass()
hasCode()
toString()

notify()
notifyAll()
wait()
wait(long timeout)
wait(long timeout, in nanos)
```

Writing Final Classes and Methods

Methods declared final cannot be overridden

```
class ChessAlgorithm {
    enum ChessPlayer { WHITE, BLACK }
    ...
    final ChessPlayer getFirstPlayer() {
        return ChessPlayer.WHITE;
    }
    ...
}
```

Classes declared final cannot be subclassed.

Abstract Classes and Methods

Abstract classes cannot be instantiated but they can be subclassed. An abstract class is a class that is declared abstract.

```
public abstract class GraphicObject {
   // declare fields
   // declare non-abstract methods
   abstract void draw();
}
```

An abstract method is a method declared without an implementation.

```
abstract void moveTo(double deltaX, double deltaY);
```

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, the subclass must also be declared abstract.

All of the methods in an interface (see the Interfaces section) are implicitly abstract.

Unlike interfaces, abstract classes can contain fields that are not static and final, and they can contain implemented methods.

Abstract classes are most commonly subclassed to share pieces of implementation.

```
abstract class GraphicObject {
    int x, y;
    ...
    void moveTo(int newX, int newY) {
        ...
    }
    abstract void draw();
    abstract void resize();
}

class Circle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}

class Rectangle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
```

Numbers

Intro

"Boxing" is when a primitive datatype is wrapped as an object. "Unboxing" is when an object is converted into a primitive datatype.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/data/numberclasses.html

The following table lists the instance methods that all the subclasses of the Number class implement.

byte byteValue()
short shortValue()
int intValue()
long longValue()
float floatValue()
double doubleValue()
int compareTo(Byte anotherByte)
int compareTo(Double anotherDouble)
int compareTo(Float anotherFloat)
int compareTo(Integer anotherInteger)
int compareTo(Long anotherLong)
int compareTo(Short anotherShort)
boolean equals(Object obj)
static Integer decode(String s)
static int parseInt(String s)
static int parseInt(String s, int radix)
String toString()
static String toString(int i)
static Integer valueOf(int i)
static Integer valueOf(String s)
static Integer valueOf(String s, int radix)

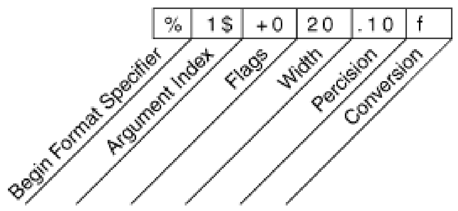
Formatting output

Can use format and printf (equivalent to each other) for number formatting, as well as print and println.

```
System.out.format("The value of " + "the float variable is " +
     "%f, while the value of the " + "integer variable is %d, " +
     "and the string is %s", floatVar, intVar, stringVar);

System.out.println(String.format("%tF %tR%n", c, c, c));
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/data/numberformat.html

Format specifiers. See (Bentley, Java Reference - Language - Printf Style Formatting.docx, 2013)



```
// General, character, a numeric data types
%[argument_index$][flags][width][.precision]conversion
// Dates and times
%[argument_index$][flags][width]conversion
```

argument_index is a decimal integer indicating the position of the argument in the argument list. The first argument is referenced by "1$", the second by "2$", etc.

flags is a set of characters that modify the output format. The set of valid flags depends on the conversion.

width is a non-negative decimal integer indicating the minimum number of characters to be written to the output

precision is a non-negative decimal integer usually used to restrict the number of characters. The specific behavior depends on the conversion.

conversion is a character indicating how the argument should be formatted. The set of valid conversions for a given argument depends on the argument's data type.

(Oracle, 2012) http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html

Table 2 Flags

Flag	General	Character	Integral	Floating Point	Date/Time	Description
'-'	y	y	y	y	y	The result will be left-justified.
'#'	y1	-	y3	y	-	The result should use a conversion-dependent alternate form
'+'	-	-	y4	y	-	The result will always include a sign
' '	-	-	y4	y	-	The result will include a leading space for positive values
'0'	-	-	y	y	-	The result will be zero-padded
','	-	-	y2	y5	-	The result will include locale-specific grouping separators
'('	-	-	y4	y5	-	The result will enclose negative numbers in parentheses

(Oracle, 2012) http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html

Table 3 Conversions in general

Conversion	Argument Category	Description

'b', 'B'	general	If the argument arg is null, then the result is "false". If arg is a boolean or Boolean, then the result is the string returned by String.valueOf(arg). Otherwise, the result is "true".
'h', 'H'	general	If the argument arg is null, then the result is "null". Otherwise, the result is obtained by invoking Integer.toHexString(arg.hashCode()).
's', 'S'	general	If the argument arg is null, then the result is "null". If arg implements Formattable, then arg.formatTo is invoked. Otherwise, the result is obtained by invoking arg.toString().
'c', 'C'	character	The result is a Unicode character
'd'	integral	The result is formatted as a decimal integer
'o'	integral	The result is formatted as an octal integer
'x', 'X'	integral	The result is formatted as a hexadecimal integer
'e', 'E'	floating point	The result is formatted as a decimal number in computerized scientific notation
'f'	floating point	The result is formatted as a decimal number
'g', 'G'	floating point	The result is formatted using computerized scientific notation or decimal format, depending on the precision and the value after rounding.
'a', 'A'	floating point	The result is formatted as a hexadecimal floating-point number with a significand and an exponent
't', 'T'	date/time	Prefix for date and time conversion characters. See Date/Time Conversions.
'%'	percent	The result is a literal '%' ('\u0025')
'%n'	line separator	The result is the platform-specific line separator

http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html

The following datetime conversions are prefixed by "t" or "T".

Table 4 Conversions, time

'H'	Hour of the day for the 24-hour clock, formatted as two digits with a leading zero as necessary i.e. 00 - 23.
'I'	Hour for the 12-hour clock, formatted as two digits with a leading zero as necessary, i.e. 01 - 12.
'k'	Hour of the day for the 24-hour clock, i.e. 0 - 23.
'l'	Hour for the 12-hour clock, i.e. 1 - 12.
'M'	Minute within the hour formatted as two digits with a leading zero as necessary, i.e. 00 - 59.
'S'	Seconds within the minute, formatted as two digits with a leading zero as necessary, i.e. 00 - 60 ("60" is a special value required to support leap seconds).
'L'	Millisecond within the second formatted as three digits with leading zeros as necessary, i.e. 000 - 999.
'N'	Nanosecond within the second, formatted as nine digits with leading zeros as necessary, i.e. 000000000 - 999999999.
'p'	Locale-specific morning or afternoon marker in lower case, e.g."am" or "pm". Use of the conversion prefix 'T' forces this output to upper case.
'z'	RFC 822 style numeric time zone offset from GMT, e.g. -0800. This value will be adjusted as necessary for Daylight Saving Time. For long, Long, and Date the time zone used is the default time zone for this instance of the Java virtual machine.
'Z'	A string representing the abbreviation for the time zone. This value will be adjusted as necessary for Daylight Saving Time. For long, Long, and Date the time zone used is the default time zone for this instance of the Java virtual machine. The Formatter's locale will supersede the locale of the argument (if any).

's'	Seconds since the beginning of the epoch starting at 1 January 1970 00:00:00 UTC, i.e. Long.MIN_VALUE/1000 to Long.MAX_VALUE/1000.
'Q'	Milliseconds since the beginning of the epoch starting at 1 January 1970 00:00:00 UTC, i.e. Long.MIN_VALUE to Long.MAX_VALUE.

(Oracle, 2012) http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html

Table 5 Conversions, dates

'B'	Locale-specific full month name, e.g. "January", "February".
'b'	Locale-specific abbreviated month name, e.g. "Jan", "Feb".
'h'	Same as 'b'.
'A'	Locale-specific full name of the day of the week, e.g. "Sunday", "Monday"
'a'	Locale-specific short name of the day of the week, e.g. "Sun", "Mon"
'C'	Four-digit year divided by 100, formatted as two digits with leading zero as necessary, i.e. 00 - 99
'Y'	Year, formatted as at least four digits with leading zeros as necessary, e.g. 0092 equals 92 CE for the Gregorian calendar.
'y'	Last two digits of the year, formatted with leading zeros as necessary, i.e. 00 - 99.
'j'	Day of year, formatted as three digits with leading zeros as necessary, e.g. 001 - 366 for the Gregorian calendar.
'm'	Month, formatted as two digits with leading zeros as necessary, i.e. 01 - 13.
'd'	Day of month, formatted as two digits with leading zeros as necessary, i.e. 01 - 31
'e'	Day of month, formatted as two digits, i.e. 1 - 31.

(Oracle, 2012) http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html

Table 6 Conversions, Datetime compounds.

'R'	Time formatted for the 24-hour clock as "%tH:%tM"
'T'	Time formatted for the 24-hour clock as "%tH:%tM:%tS".
'r'	Time formatted for the 12-hour clock as "%tI:%tM:%tS %Tp". The location of the morning or afternoon marker ('%Tp') may be locale-dependent.
'D'	Date formatted as "%tm/%td/%ty".
'F'	ISO 8601 complete date formatted as "%tY-%tm-%td".
'c'	Date and time formatted as "%ta %tb %td %tT %tZ %tY", e.g. "Sun Jul 20 16:17:00 EDT 1969".

(Oracle, 2012) http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html

Digit examples

```
long n = 461012;
long x = -461012;

// "461012"
System.out.format("%d%n", n);

// "00461012"
System.out.format("%08d%n", n);

// " +461012"
System.out.format("%+8d%n", n);

// "-461,012  |"
System.out.format("%- ,10d|%n", n);

// "-461,012  |"
System.out.format("%-,10d|%n", x);

// "(461,012) |"
```

```
System.out.format("%(-,10d|%n", x);

// "+461,012"
System.out.format("%+,8d%n%n", n);
```

Float examples

```
double pi = Math.PI;

// "3.141593"
System.out.format("%f%n", pi);

// "3.142"
System.out.format("%.3f%n", pi);

// "     3.142"
System.out.format("%10.3f%n", pi);

// "3.142"
System.out.format("%-10.3f%n", pi);

// "3,1416"
System.out.format(Locale.FRANCE,
                 "%-10.4f%n%n", pi);
```

Date Time Examples

```
Calendar c = Calendar.getInstance();

// "May 29, 2006"
System.out.format("%tB %te, %tY%n", c, c, c);

// "2:34 am"
System.out.format("%tl:%tM %tp%n", c, c, c);

// "05/29/06"
System.out.format("%tD%n", c);

// "2012-06-12"
System.out.format("%tY-%tm-%td%n", c, c, c);

// "2012-06-12 17:47"
System.out.format("%tF %tR%n", c, c);

// "2012-06-12 17:47"
System.out.println(String.format("%1$tF %1$tR%n", c));
```

DecimalFormat

```
import java.text.DecimalFormat;

public class DecimalFormatDemo {

  static public void customFormat(String pattern, double value) {
    DecimalFormat myFormatter = new DecimalFormat(pattern);
    String output = myFormatter.format(value);
    System.out.println(value + "  " + pattern + "  " + output);
  }

  static public void start() {
    // 123,456.789
    customFormat("###,###.###", 123456.789);

    // 123456.79
    customFormat("###.##", 123456.789);

    // 000123.780
    customFormat("000000.000", 123.78);

    // $12,345.67
    customFormat("$###,###.###", 12345.67);
  }
}
```

Table 7 DecimalFormat Pattern Characters

Symbol	Location	Meaning
0	Number	Digit
#	Number	Digit, zero shows as absent
.	Number	Decimal separator or monetary decimal separator
-	Number	Minus sign
,	Number	Grouping separator
E	Number	Separates mantissa and exponent in scientific notation. Need not be quoted in prefix or suffix.
;	Subpattern boundary	Separates positive and negative subpatterns
%	Prefix or suffix	Multiply by 100 and show as percentage
\u2030	Prefix or suffix	Multiply by 1000 and show as per mille value
¤ (\u00A4)	Prefix or suffix	Currency sign, replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator.
'	Prefix or suffix	Used to quote special characters in a prefix or suffix, for example, "'#'#" formats 123 to "#123". To create a single quote itself, use two in a row: "# o''clock".

(Oracle, 2012) http://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

java.lang.Math

(Oracle, 2012) http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

All methods in java.lang.Math are static. Call methods like this …

```
// Qualified
Math.cos(angle);

// Unqualified
import static java.lang.Math.*;
cos(angle);
```

The two java.lang.Math constants ought be qualified when called, to disambiguate them (e.g there is a java.lang.StrictPath.PI constant).

```
Math.E;
Math.PI
```

Table 8 Math rounding methods

Method	Description
double abs(double d) float abs(float f) int abs(int i) long abs(long lng)	Returns the absolute value of the argument.
double ceil(double d)	Returns the smallest integer that is greater than or equal to the argument. Returned as a double.
double floor(double d)	Returns the largest integer that is less than or equal to the argument. Returned as a double.

double rint(double d)	Returns the integer that is closest in value to the argument. Returned as a double.
long round(double d) int round(float f)	Returns the closest long or int, as indicated by the method's return type, to the argument.
double min(double arg1, double arg2) float min(float arg1, float arg2) int min(int arg1, int arg2) long min(long arg1, long arg2)	Returns the smaller of the two arguments.
double max(double arg1, double arg2) float max(float arg1, float arg2) int max(int arg1, int arg2) long max(long arg1, long arg2)	Returns the larger of the two arguments.

Table 9 Math exponential and logarithmic methods

Method	**Description**
double exp(double d)	Returns the base of the natural logarithms, e, to the power of the argument.
double log(double d)	Returns the natural logarithm of the argument.
double pow(double base, double exponent)	Returns the value of the first argument raised to the power of the second argument.
double sqrt(double d)	Returns the square root of the argument.

Table 10 Math Trig methods

Method	**Description**
double sin(double d)	Returns the sine of the specified double value.
double cos(double d)	Returns the cosine of the specified double value.
double tan(double d)	Returns the tangent of the specified double value.
double asin(double d)	Returns the arcsine of the specified double value.
double acos(double d)	Returns the arccosine of the specified double value.
double atan(double d)	Returns the arctangent of the specified double value.
double atan2(double y, double x)	Converts rectangular coordinates (x, y) to polar coordinate (r, theta) and returns theta.
double toDegrees(double d) double toRadians(double d)	Converts the argument to degrees or radians.

Random

```
/**
 * @param min
 *   The lower bound of the range, inclusive.
 * @param max
 *   The upper bound of the range, inclusive.
 * @return.
 *  An random integer between min and max (inclusive).
 */
private static int RandomInteger(int min, int max) {
  return (int) Math.round(Math.random() * ((max - min) + 1)) + min;
}
```

There is also java.util.Random

Character Object

Java sometimes needs to work with an object rather than the primitive datatype. So a character is sometimes "boxed", converted to an object, or "unboxed", converted back to a primitive datatype.

```
// Create a character object
Character ch = new Character('a');
```

Table 11 Character class methods (some of them)

Method	Description
boolean isLetter(char ch) boolean isDigit(char ch)	Determines whether the specified char value is a letter or a digit, respectively.
boolean isWhitespace(char ch)	Determines whether the specified char value is white space.
boolean isUpperCase(char ch) boolean isLowerCase(char ch)	Determines whether the specified char value is uppercase or lowercase, respectively.
char toUpperCase(char ch) char toLowerCase(char ch)	Returns the uppercase or lowercase form of the specified char value.
toString(char ch)	Returns a String object representing the specified character value — that is, a one-character string.

Escape Sequences

To escape a character proceed it with a backslash (\).

```
System.out.println("She said \"Hello!\" to me.");
```

Table 12 Escape sequences

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formfeed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.
\uXXXX (hexadecimal)	Unicode Code Point. E.g. " \u03B2" is Greek small letter Beta, "β"

Strings

Basics

In Java strings are objects.

Create string via string literal.

```
String greeting = "Hello world!";
```

Create string via object techniques.

```
char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
String helloString = new String(helloArray);
```

The character and string classes are immutable. Once created the object cannot change.

Class (static) methods that return a string object just create a new object.

String length.

```
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();
```

Concatenation

Concatenating Strings.

```
// + operator
"Hi, " + "there";

// concat() method
"Hi, ".concat("there");
string1.concat(string2);
```

Split string in Code

Splitting string literals over two lines: use a plus (+) sign at the end of the previous line or the beginning of the next line.

```
String quote = "Now is the time for all" +
               " good men";
```

Printf-style formatting

See (Bentley, Java Reference - Language - Printf Style Formatting.docx, 2013)

Format strings (String.format) can be reused.

```
String fs;
fs = String.format("The value of the float " +
                   "variable is %f, while " +
                   "the value of the " +
                   "integer variable is %d, " +
                   " and the string is %s",
                   floatVar, intVar, stringVar);
System.out.println(fs);
```

But you can use the printf() or format() method of System.out.printf

```
System.out.printf("The value of the float " +
                  "variable is %f, while " +
                  "the value of the " +
                  "integer variable is %d, " +
                  "and the string is %s",
                  floatVar, intVar, stringVar);
```

Regular Expressions

See (Bentley, Java Reference - Language - Regular Expressions.docx, 2013)

Conceptual overview example.

```
// The regular expression (or "regex" or "pattern")
"(\w{3})(dog)# Comment"

// final static int flags = Pattern.CASE_INSENSITIVE | Pattern.COMMENTS;

// String to search (or "input string")
"catdogpussyratdog"

// Matches and captures
Match (1) of text "catdog" starting at index 0 and ending at 6.
      Capture groups: 2
      Capture group (0) of text "catdog" starting at index 0 and ending at 6.
      Capture group (1) of text "cat" starting at index 0 and ending at 3.
      Capture group (2) of text "dog" starting at index 3 and ending at 6.
Match (2) of text "ratdog" starting at index 11 and ending at 17.
      Capture groups: 2
      Capture group (0) of text "ratdog" starting at index 11 and ending at 17.
      Capture group (1) of text "rat" starting at index 11 and ending at 14.
      Capture group (2) of text "dog" starting at index 14 and ending at 17.

// Replacement String
"@$2cool$1@"

// Replacement Result
@dogcoolcat@pussy@dogcoolrat@
```

Single line operation examples.

```
final static String regex = "(\\w{3})(dog)";
final static String stringToSearch = "catdogpussycatdog";
final static String replacement = "$2cool$1";

private static void matches() {

  // Boolean match
  // Does the regex match the whole stringToSearch?
  System.out.printf("\"%s\" matches the whole of \"%s\"?: %s%n", regex,
      stringToSearch, stringToSearch.matches(regex));
  // "(\w{3})(dog)" matches the whole of "catdogpussycatdog"?: false

  // Does the regex match part of the stringToSearch?
  System.out.printf("\"%s\" matches part of \"%s\"?: %s%n", regex, stringToSearch,
      Pattern.compile(regex).matcher(stringToSearch).find());
  // "(\w{3})(dog)" matches part of "catdogpussycatdog"?: true

} // matches()

private static void replacement() {

  // Substitute the matches (java.lang.String)
  System.out.printf("Replace all \"%s\" matches of \"%s\" with \"%s\": %s%n",
      regex, stringToSearch, replacement,
      stringToSearch.replaceAll(regex, replacement));
  // Replace all "(\w{3})(dog)" matches of "catdogpussycatdog" with "$2cool$1":
  // dogcoolcatpussydogcoolcat
  // Substitute the matches (java.util.regex)
  System.out.printf("Replace all \"%s\" matches of \"%s\" with \"%s\": %s%n",
      regex, stringToSearch, replacement,
      Pattern.compile(regex).matcher(stringToSearch).replaceAll(replacement));
  // Replace all "(\w{3})(dog)" matches of "catdogpussycatdog" with "$2cool$1":
  // dogcoolcatpussydogcoolcat

} // replacement
```

(Bentley, TutorialAtOracle Code Examples - Regex, 2013) RegexDemo.java &
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/regex/matcher.html

Conversion

Convert strings to numbers, using .valueOf and …Value() of a number subclass that wraps a primitive numeric type (Byte, Integer, Double, Float, Long, and Short). More direct technique is to use .parseFloat()

```
String strX = "34.56";
String strY = "-789.34";
float x = Float.valueOf(strX).floatValue();

// Preferred (more direct)
float y = Float.parseFloat(strY);

System.out.println("x + y = " + (x + y));
```

Convert numbers to strings.

```
// Concatenate empty string
int i;
String str = "" + i;

// valueOf class method
String str = String.valueOf(i);

// Number subclass toString()
int i;
double d;
String strA = Integer.toString(i);
String strB = Double.toString(d);
```

Manipulation

String methods

Get characters and substrings by index.

```
// charAt()
String anotherPalindrome = "Niagara. O roar again!";
char aChar = anotherPalindrome.charAt(9);
// 'O'

// substring();
String anotherPalindrome = "Niagara. O roar again!";
String roar = anotherPalindrome.substring(11, 15);
// "roar"
```

Table 13 substring()

Method	**Description**

String substring(int beginIndex, int endIndex)	Returns a new string that is a substring of this string. The first integer argument specifies the index of the first character. The second integer argument is the index of the last character - 1.
String substring(int beginIndex)	Returns a new string that is a substring of this string. The integer argument specifies the index of the first character. Here, the returned substring extends to the end of the original string.

Table 14 Other manipulation

Method	**Description**
String[] split(String regex) String[] split(String regex, int limit)	Searches for a match as specified by the string argument (which contains a regular expression) and splits this string into an array of strings accordingly. The optional integer argument specifies the maximum size of the returned array. Regular expressions are covered in the lesson titled "Regular Expressions."
CharSequence subSequence(int beginIndex, int endIndex)	Returns a new character sequence constructed from beginIndex index up until endIndex - 1.
String trim()	Returns a copy of this string with leading and trailing white space removed.
String toLowerCase() String toUpperCase()	Returns a copy of this string converted to lowercase or uppercase. If no conversions are necessary, these methods return the original string.

Table 15 Search

Method	**Description**
int indexOf(int ch) int lastIndexOf(int ch)	Returns the index of the first (last) occurrence of the specified character.
int indexOf(int ch, int fromIndex) int lastIndexOf(int ch, int fromIndex)	Returns the index of the first (last) occurrence of the specified character, searching forward (backward) from the specified index.
int indexOf(String str) int lastIndexOf(String str)	Returns the index of the first (last) occurrence of the specified substring.
int indexOf(String str, int fromIndex) int lastIndexOf(String str, int fromIndex)	Returns the index of the first (last) occurrence of the specified substring, searching forward (backward) from the specified index.
boolean contains(CharSequence s)	Returns true if the string contains the specified character sequence.

```
String pathSlashFileName = "C:\\Data\\Temp\\Nice.txt";
int dotIndex = pathSlashFileName.lastIndexOf('.');
String extension = pathSlashFileName.substring(dotIndex);
System.out.println(extension);

// ".txt"
```

Table 16 Replace

Method	**Description**

String replace(char oldChar, char newChar)	Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
String replace(CharSequence target, CharSequence replacement)	Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.
String replaceAll(String regex, String replacement)	Replaces each substring of this string that matches the given regular expression with the given replacement.
String replaceFirst(String regex, String replacement)	Replaces the first substring of this string that matches the given regular expression with the given replacement.

Table 17 Comparison

Method
boolean endsWith(String suffix) boolean startsWith(String prefix)
boolean startsWith(String prefix, int offset)
int compareTo(String anotherString)
int compareToIgnoreCase(String str)
boolean equals(Object anObject)
boolean equalsIgnoreCase(String anotherString)
boolean regionMatches(int toffset, String other, int ooffset, int len)
boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
boolean matches(String regex)

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/data/comparestrings.html

String, StringBuilder and StringBuffer

Class	**Purpose**
String	For strings that don't change.
StringBuilder	For strings that do change, in single threaded operations. (not thread safe).
StringBuffer	For strings that do change, in multithreaded operations. (Thread safe)

```
// Reverse it
String palindrome = "Dot saw I was Tod";

StringBuilder sb = new StringBuilder(palindrome);
sb.reverse(); // reverse it
System.out.println(sb);

...
// Allocate length without an initial string
String palindrome = "Dot saw I was Tod";

StringBuilder sb = new StringBuilder();
sb.setLength(palindrome.length());

sb.setCharAt(0, '^');
System.out.println(sb);
```

(Oracle, 2011) http://docs.oracle.com/javase/6/docs/api/java/lang/StringBuilder.html

Unicode

Unicode Tools

Obtain unicode characters, both their literal representation and their code, through windows charmap.

- Windows Key > Charmap > Font: Lucida Sans Unicode.
- Advanced View: Ticked.
- Character Set: Unicode; Group By: Unicode Subrange.
- Choose Subrange.

Eclipse Settings

To display literal unicode characters in eclipse source code.

- Eclipse > File > Properties > Resource > Text file encoding > Other: UTF-8

To display literal unicode characters in the eclipse console.

- Select the Project.
- Run (Menu) > Debug Configurations > Java Application > [Choose Project].
- Common (tab) > Encoding > Other: UTF-8.

Ian Bull, 2013-Feb-21, "Pro Tip: Unicode characters in the Eclipse console", http://eclipsesource.com/blogs/2013/02/21/pro-tip-unicode-characters-in-the-eclipse-console/

Overview

See also List of Unicode characters

(Wikipedia) http://en.wikipedia.org/wiki/List_of_Unicode_characters

Unicode characters are represented literally or as by their Unicode Code Point in hexadecimal.

```
System.out.println("Hello 利\u5229 Greek \u03B2");

// Output:
Hello 利利 Greek β
```

Unicode doesn't say how characters are represented in bytes. It just assigns random numbers to characters.

(Mansoor, 2013)

Unicode Code Points, in hexadecimal, have two named ranges:

- Basic Multilingual Plane: U+0000 to U+FFFF (65535) {16 bits}; and
- Supplementary characters: U+10000 (65536) to U+10FFFF (1,114,111) {32 bits}

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/terminology.html

A **character set** is a set of characters. One set of characters might be used for multiple languages. For example, the Latin character set is used by English and most European languages.

```
A,
B,
C,
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/terminology.html

A **coded character set** is a character set for which each character has been assigned a number.

```
A 65,
B 66,
C 67,

(Numbers in decimal)
```

http://en.wikipedia.org/wiki/Character_encoding

A **code point** is the number which maps to a character in a coded character set.

```
65,
66,
67
```

http://en.wikipedia.org/wiki/Character_encoding
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/terminology.html

A **character encoding form** specifies the conversion of **code points**, numbers which map to a character in a coded character set, to limited-size integer **code values** that facilitate storage in a system that represents numbers in binary form using a fixed number of bits.

Character	Code Point (Decimal)	Code Point (Hexadecimal)	Code Point (Binary)	UTF-8 (encoding) Code Value in bits	UTF-8 (encoding) Code Value in hex
$	36	U+0024	00100100	00100100	24
A	65	U+0041	01000001	01000001	41
B	66	U+0042	01000010	01000010	42
C	67	U+0043	01000011	01000011	43
𤭢	150370	U+24B62	00000010 01001011 01100010	11110000 10100100 10101101 10100010	F0 A4 AD A2

> The simplest CEF system is simply to choose large enough units that the values from the coded character set can be encoded directly (one code point to one code value). ... However, as the size of the coded character set increases (e.g. modern Unicode requires at least 21 bits/character), this becomes less and less efficient, and it is difficult to adapt existing systems to use larger code values. Therefore, most systems working with later versions of Unicode use either UTF-8, which maps Unicode code points to variable-length sequences of octets, or UTF-16, which maps Unicode code points to variable-length sequences of 16-bit words.
>
> *http://en.wikipedia.org/wiki/Character_encoding*

http://en.wikipedia.org/wiki/Character_encoding
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/terminology.html

In java a char datatype is implemented with 16 bits. Unicode characters in the Basic Multilingual Plane are thus represented with a single char. Unicode supplementary characters are coded with two chars, called *surrogates*, providing 32 bits. That is, java represents Unicode characters using UTF-16.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/terminology.html
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/supplementaryChars.html

A **user character** may be composed of more than one Unicode character.

```
// the user character ü can be composed by combining the Unicode characters \u0075 (u) and
\u00a8 (¨). This isn't the best example, however, because the character ü may also be
represented by the single Unicode character \u00fc.


// In Arabic the word for house is:
بَيْٺُ

/// This word contains three user characters, but it is composed of the following six
Unicode characters:

String house = "\u0628" + "\u064e" + "\u064a" + "\u0652" + "\u067a" + "\u064f";
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/char.html

Unicode methods in the Character and String class

Character conversion (between char and code points) method examples.

```
// Provide Code Point in hexidecimal
Character.toChars(0x24B62); -> 𤭢

// Provide character
Character.codePointAt("𤭢", 0);

// Returns Code Point in Decimal
150370
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) UnicodeDemo.java
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/characterClass.html

String, StringBuffer, and StringBuilder have constructors and methods that work with Unicode supplementary characters (as above).

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/characterClass.html

Create a string from a code point.

```
/**
 * Handles Basic Multilingual Plane and Supplementary characters.
 * @param codePoint
 *  codePoint in decimal or hexadecimal. E.g. 65, 0x41
 * @return
 *  The string that the codePoint represents. E.g "A" or "𤭢"
 */
private static String newStringFromCodePoint(int codePoint) {
    return new String (Character.toChars(codePoint));
}

private static void testNewStringFromCodePoint() {
  System.out.println(newStringFromCodePoint(65));
  System.out.println(newStringFromCodePoint(0x41));
  System.out.println(newStringFromCodePoint(150370));
  System.out.println(newStringFromCodePoint(0x24B62));
}
```

```
// Output
A
A
𤭢
𤭢
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) UnicodeDemo.java
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/usage.html

Unicode methods, traps to avoid

Character verification method examples

```
Character.isValidCodePoint(156); -> true
Character.isValidCodePoint(0x10FFFF); -> true
Character.isValidCodePoint(0x110000); -> false
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) UnicodeDemo.java
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/characterClass.html
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/design.html

To work with Unicode supplementary characters don't use verification methods that take a char datatype, use those that take a code point.

```
// Don't do this
Character.isLowerCase('A'); -> false

// Do this
Character.isLowerCase(Character.codePointAt("A", 0)); -> false
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) UnicodeDemo.java
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/characterClass.html
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/design.html

The java tutoral advice is to avoid using string.length() and use, instead, string.codePointCount(...) in order to correctly count supplementary characters. However, no difference was found by experimentation. Maybe "length" has been update to handle supplementary characters???

```
private static void countingCharacters() {
  String[] strings = { "Hello ", "Hello 利\u5229ßx" };
  for (String stringLoop : strings) {
    System.out.printf("\"%s\":%n", stringLoop);
    System.out.printf(" length(): %d%n", stringLoop.length());
    System.out.printf(" codePointCount(...): %d%n",
        stringLoop.codePointCount(0, stringLoop.length()));
    System.out.println();
  }
}

// Output
"Hello ":
 length(): 6
 codePointCount(...): 6

"Hello 利利ßx":
 length(): 10
 codePointCount(...): 10  // Same as with length().
```

(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) UnicodeDemo.java
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/characterClass.html

When converting case use String methods not Character methods. Character methods choke on some characters (e.g. "The lowercase German character ß, for example, becomes two characters, SS, when converted to uppercase").

```
// Don't use
Character.toUpperCase(int codePoint)
Character.toLowerCase(int codePoint)

// Use these
String.toUpperCase(int codePoint)
String.toLowerCase(int codePoint)
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/design.html

Be careful when deleting characters.

> When invoking the StringBuilder.deleteCharAt(int index) or StringBuffer.deleteCharAt(int index) methods where the index points to a supplementary character, only the first half of that character (the first `char` value) is removed. First, invoke the Character.charCount method on the character to determine if one or two `char` values must be removed.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/design.html

Be careful when reversing characters.

> When invoking the StringBuffer.reverse() or StringBuilder.reverse() methods on text that contains supplementary characters, the high and low surrogate pairs are reversed which results in incorrect and possibly invalid surrogate pairs.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/design.html

Printf style output

Printf style output can handle Unicode code points expressed in decimal or hexadecimal.

```
      System.out.printf("%c", 'h');  |              "h" | Character
      System.out.printf("%c", 104);  |              "h" | Character
     System.out.printf("%c", 0x68);  |              "h" | Character
   System.out.printf("%c", 150370);  |             "𤭢" | Character
```

(Bentley, Java Reference - Language - Printf Style Formatting.docx, 2013)
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/usage.html

Converting Non-Unicode Text

There exists some apparently unnecessary techniques for converting non latin characters (expressed as a hexadecimal) into their Unicode representation; and converting between character encodings in the Java Tutorials under "Converting Non-Unicode Text".

(Oracle, 2011) http://docs.oracle.com/javase/tutorial/i18n/text/convertintro.html

Normalizing Text

Normalizing text is transforming text to make it useable in a way that it might not have been before (e.g. for searching or sorting). Normalization might be needed, for example, to: convert characters with diacritical marks; change case; decompose ligatures; or convert half-width katakana characters to full-width characters.

```
private static void NormalizationDemo() {
  String word = "schön";

  Normalizer.Form [] normalizerForms = { Normalizer.Form.NFC, Normalizer.Form.NFD,
      Normalizer.Form.NFKC, Normalizer.Form.NFKD };

  System.out.printf("Original: %s%n", word);
  for (Normalizer.Form normalizerFormLoop : normalizerForms) {
     String normalizedWord = Normalizer.normalize(word, normalizerFormLoop);
     System.out.printf("%8s: %s%n", normalizerFormLoop.name(), normalizedWord);
  }
}

// Output
Original: schön
     NFC: schön
     NFD: schön
    NFKC: schön
    NFKD: schön

// Java Tutorial claims output would be
Original: schön
     NFC: schön
     NFD: scho\u0308n
    NFKC: schön
    NFKD: scho\u0308n
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/normalizerapi.html
(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) UnicodeDemo.java

BiDirectional Text

If you need to handle bidirectional text read da stuff at ...

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/text/bidi.html

Internationalization of URLs

To handle the internationalisation of urls use methods from java.net.IDN.

```
private static void internationalisationOfUrlsDemo() {
  String nonAsciiUrl = "http://清华大学.cn";

  System.out.println(IDN.toASCII(nonAsciiUrl));
  System.out.println(IDN.toUnicode("xn--http://-qp2lt84bkofex3d.cn"));
}

// Output
xn--http://-qp2lt84bkofex3d.cn
http://清华大学.cn
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/i18n/network/idn.html
(Bentley, TutorialAtOracle Code Examples - HelloWorld, 2013) InternationalisationOfUrls.java

Generics

Generics allow you to restrict objects so that programming errors can be caught at compile time rather than run time.

Generics also allow you to reuse code instead of having to write several overloaded methods that do the same thing. (thenewboston, 2010)

Generic Types aka "Generic classes" (and interfaces)

A "generic type" is a generic class or interface that is parameterized over types.

Define a generic class with one or more type parameters (aka "type variables").

```
public class Box<T> {

  private T t;

  public void set(T t) {
    this.t = t;
  }

  public T get() {
    return this.t;
  }
}
```

Use class with a generic type

```
public static void start() {
  // We are passing a "type argument" to the generic class.
  Box<Integer> integerBox = new Box<Integer>();

      // In Java SE7 and above the following form is legal
      Box<Integer> integerBox = new Box<>();

  // Correct
  integerBox.set(new Integer(10));

  // Compile error:
  // integerBox.set("10");

  Integer x = integerBox.get();
  System.out.println(x);
}
```

Table 18 Generics, formal type parameters naming convention

E	**Element**
K	Key
V	Value
N	Number
T	Type
S,U,V etc.	2nd, 3rd, 4th types

Another example to demonstrate the utility of generics

```
public interface IPair<K, V> {
  public K getKey();
  public V getValue();
}

public class OrderedPair<K, V> implements IPair<K, V> {

  private K key;
  private V value;

  public OrderedPair(K key, V value) {
    this.key = key;
    this.value = value;
  }

  public K getKey() {
    return key;
  }

  public V getValue() {
    return value;
  }

  public String toString() {
    return "Key: " + this.getKey() + " Value: " + this.getValue();
  }
}

OrderedPair<String, Integer> p1 = new OrderedPair<String, Integer>("Even", 8);
OrderedPair<String, String>  p2 = new OrderedPair<String, String>("hello", "world");
```

Raw types are a legacy thing

```
// Normal declaration and creation of a generic type class
Box<Integer> integerBox = new Box<Integer>();

// Legacy method using a "raw" type (using the same generic class definition).
Box rawBox = Box();
```

Generic methods

Generic methods defined and used example, basic.

```
public class Util {
    // Generic static method
    public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {
        return p1.getKey().equals(p2.getKey()) &&
               p1.getValue().equals(p2.getValue());
    }
}

Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");
boolean same = Util.<Integer, String>compare(p1, p2);
```

Generic method defined and used example, when returning a type specified with a generic.

```
// Define generic method (the example is contrived)
static private <E> List<E> heapSort(Collection<E> c) {
    Queue<E> queue = new PriorityQueue<E>(c);
    System.out.printf("%s PriorityQueue%n", queue);
    List<E> resultList = new ArrayList<E>();

    while (!queue.isEmpty()) {
        resultList.add(queue.remove());
    }

    return resultList;
}

// Use generic method
private static List<String> mList = Arrays.asList("Xeno",
                                                   "Marx",
                                                   "Alphie",
                                                   "Zara",
                                                   "Marx",
                                                   "Marx",
                                                   "Hasselhoff",
                                                   "Beiber");

static private void priorityQueueDemo() {
    List<String> resultList = null;
    System.out.println(mList);
    resultList = heapSort(mList);
    System.out.println(resultList);
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/collections/interfaces/queue.html
C:\Users\John\Documents\Sda\Code\Java\Examples\TutorialAtOracle\HelloWorld\src\helloworld\demo\collections\QueueDemo.java

Type inference allows you to omit the types when calling (but not recommended)

```
boolean same = Util.compare(p1, p2);
```

Bounded Type Parameters

Restrict the types that can be passed by creating an “upper bound” with the extends keyword. This is a “bounded type parameter”.

```
// This example uses a generic method …
public static <U extends Number> void inspect (U u) {
  System.out.println("U class name: " + u.getClass().getName());
  System.out.println("U value: " + u);
}

// code executed.
inspect(new Double(10));

// causes compile error. That is the desired effect: restricting types that can be passed.
inspect("sdkflj");

// When compile error commented out and program run, the output:
U class name: java.lang.Double
U value: 10.0
```

You can use the methods defined by the upper bound (in the following example from “Number”, not “Integer”).

```
// This example uses a generic class …
public class NaturalNumber<T extends Number> {
  private T n;

  public NaturalNumber(T n)  { this.n = n; }

  public boolean isEven() {
      return n.intValue() % 2 == 0; // Using a method from "Number".
  }
}

// Execute
NaturalNumber<Integer> integerNaturalNumber = new NaturalNumber<Integer>(11);
System.out.println("IsEven " + integerNaturalNumber.isEven());

// output
IsEven false
```

You can have multiple bounds. A type variable with a multiple bounds is a subtype of all the bounds.

```
public class ComparableNumber<T extends Number & Comparable<T>>  {
  private T n;

  public ComparableNumber(T n)  { this.n = n; }

  public T getN() {
    return n;
  }
}
```

If one of the bounds, in a multiple bound declaration, is a class, it must come first.

Generic algorithms are served well by bounded type parameters.

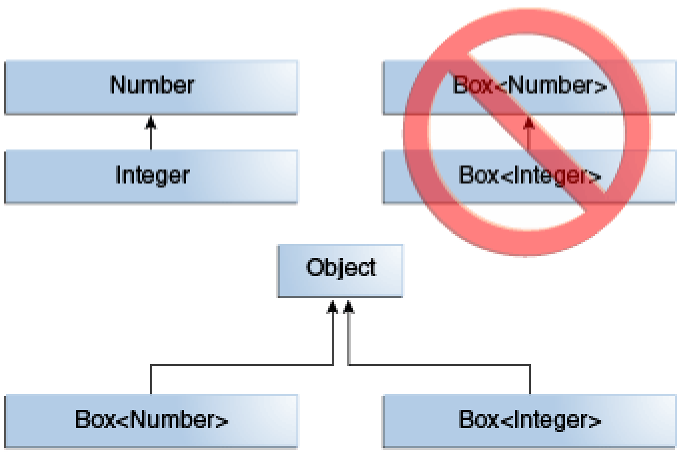
```
public static <T extends Comparable<T>> int countGreaterThan(T[] anArray, T elem) {
    int count = 0;
    for (T e : anArray)
        if (e.compareTo(elem) > 0)
            ++count;
    return count;
}
```

(Oracle, 2012) , "Generic Methods and Bounded Type Parameters", http://docs.oracle.com/javase/tutorial/java/generics/boundedTypeParams.html

Generics, Inheritance, and Subtypes

Given two concrete types A and B (for example, Number and Integer), MyClass<A> has no relationship to MyClass<B>, regardless of whether or not A and B are related. The common parent of MyClass<A> and MyClass<B> is Object.

```
Box<Integer> is not a subtype of Box<Number> even though Integer is a subtype of Number.
```



You can subtype a generic class or interface. Do this with the extends or implements keyword. When doing so, the type argument must not vary.

Using the Collections classes as an example, ArrayList<E> **implements** List<E>, and List<E> **extends** Collection<E>. So ArrayList<String> is a subtype of List<String>, which is a subtype of Collection<String>

```
// An interface example
interface PayloadList<E,P> extends List<E> {
  void setPayload(int index, P val);
  ...
}

// The following parameterizations of PayloadList are subtypes of List<String>:
```

```
PayloadList<String,String>
PayloadList<String,Integer>
PayloadList<String,Exception>
```

Type Inference

Type inference is a Java compiler's ability to look at each method invocation and corresponding declaration to determine the type argument (or arguments) that make the invocation applicable. The inference algorithm determines the types of the arguments and, if available, the type that the result is being assigned, or returned. Finally, the inference algorithm tries to find the most specific type that works with all of the arguments.

```
// To illustrate this last point, in the following example, inference determines
// that the second argument being passed to the pick method is of type Serializable:
static <T> T pick(T a1, T a2) { return a2; }
Serializable s = pick("d", new ArrayList<String>());
```

Constructors can take a generic variable that is different from the class itself

```
class MyClass<X> {
  <T> MyClass(T t) {
    // ...
  }
}

MyClass<Integer> myObject = new MyClass<>("");
```

Wildcards

The wildcard, ?, represents an unknown type.

You can use an *upper bounded wildcard* to relax the restrictions on a variable. To declare an upper-bounded wildcard, use the wildcard character ('?'), followed by the extends keyword, followed by its *upper bound.*

```
// Upperbound wildcard in generic method
public static double sumOfList(List<? extends Number> list) {
  double s = 0.0;
  for (Number n : list) {
    s += n.doubleValue();  // doubleValue is from Number.
  }
  return s;
}

// Outputs "sum = 6.0"
List<Integer> integerlist = Arrays.asList(1, 2, 3);
System.out.println("sum = " + sumOfList(integerlist));

// Outputs "sum = 7.0"
List<Double> doubleList = Arrays.asList(1.2, 2.3, 3.5);
System.out.println("sum = " + sumOfList(doubleList));
```

The term List<Number> is more restrictive than List<? extends Number> because the former matches a list of type Number only, whereas the latter matches a list of type Number or any of its subclasses.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/upperBounded.html

Use an *unbounded* wildcard, a "list of unknown type", when you are writing a method: that uses functionality of the object class; or the method in the generic class does not depend on the type parameter.

```
public static void printList(List<?> list) {
    for (Object elem: list)
        System.out.print(elem + " ");
    System.out.println();
}

// Because for any concrete type A, List<A> is a subtype of List<?>, you can use printList
to print a list of any type:
List<Integer> listInteger = Arrays.asList(1, 2, 3);
List<String>  listString = Arrays.asList("one", "two", "three");
printList(listInteger);
printList(listString);
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/unboundedWildcards.html

Use a *lower bound* wildcard to restrict the unknown type to be a specific type or its super type. Use <? super T>

```
public static void addNumbers(List<? super Integer> list) {
    for (int i = 1; i <= 10; i++) {
        list.add(i);
    }
}

// The term List<Integer> is more restrictive than List<? super Integer> because the former
matches a list of type Integer only, whereas the latter matches a list of any type that is a
supertype of Integer. So the method works on work on List<Integer>, List<Number>, and
List<Object> — anything that can hold Integer values.
```

Guidelines for when to define a parameter as an upper bounded, unbounded, or a lower bounded wildcard:

- An "in" variable is defined with an upper bounded wildcard, using the extends keyword.
- An "out" variable is defined with a lower bounded wildcard, using the super keyword.
- In the case where the "in" variable can be accessed using methods defined in the Object class, use an unbounded wildcard.
- In the case where the code needs to access the variable as both an "in" and an "out" variable, do not use a wildcard.

Using a wildcard as a return type should be avoided.

Otherwise it forces programmers using the code to deal with wildcards.

Guideline mnemonic for wildcard definitions: PECS. Producer ("in") extends; consumer ("out") super.

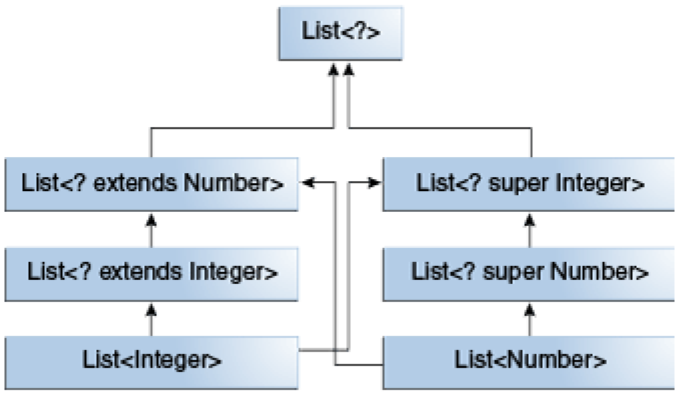
(ploygenelubricants, 2010), quoting "Effective Java 2nd Edition, Item 28: Use bounded wildcards to increase API flexibility"

Use lower and upper bounded wildcards when declaring variables to create a relationship between generic classes (or interfaces).

```
// OK. List<? extends Integer> is a subtype of List<? extends Number>
List<? extends Integer> intList = new ArrayList<Integer>();
List<? extends Number>  numList = intList;
```

```
// the code can access Number's methods through List<Integer>'s elements.

// Compile error. No relationship between List<Integer> and List<Number>
// even though there is between integer and number.
List<Integer> intList = new ArrayList<Integer>();
List<Number>  numList = intList;
```

The following diagram shows the relationships between several `List` classes declared with both upper and lower bounded wildcards.



"Wildcard Capture" is type inference with respect to wildcards.

```
// a list may be defined as List<?> but, when evaluating an expression, the compiler infers
a particular type from the code.
```

Sometimes when wildcard capture produces a compile error you can correct this with a *private helper method*.

```
import java.util.List;

// On its own: compile error.
public class WildcardError {
    void foo(List<?> i) {
        i.set(0, i.get(0));
    }
}

//  the compiler processes the i input parameter as being of type Object. When the foo
method invokes List.set(int, E), the compiler is not able to confirm the type of object that
is being inserted into the list, and an error is produced. When this type of error occurs it
typically means that the compiler believes that you are assigning the wrong type to a
variable.

// Fixed with a private helper method
public class WildcardFixed {
    void foo(List<?> i) {
        fooHelper(i);
    }

    // Helper method created so that the wildcard can be captured
    // through type inference.
    private <T> void fooHelper(List<T> c) {
        c.set(0, c.get(0));
    }
}
```

Bounded Type Parameters V Bounded Wildcards

Use a bounded type parameter if:

- You need to refer to the type parameter again;
- You need the type parameter to have multiple bounds.

(ploygenelubricants, 2010)

Use a bounded wildcard if:

- You need a lower bound.

(ploygenelubricants, 2010)

Type Erasure

Type erasure is compile time translation of generic types into ordinary code. It entails: substituting ordinary classes, interfaces, and methods for generic types; type casts; creating bridge methods.

When substituting generic types if the type parameter is bounded then the first bound is substituted.

```
// Source is bounded
public class Node<T extends Comparable<T>> {
    private T data;
    private Node<T> next;

    public Node(T data, Node<T> next) {
        this.data = data;
        this.next = next;
    }

    public T getData() { return data; }
}

// Erasure substitution wtih first bound
public class Node {
    private Comparable data;
    private Node next;

    public Node(Comparable data, Node next) {
        this.data = data;
        this.next = next;
    }

    public Comparable getData() { return data; }
}
```

When substituting generic types if the type parameter is unbounded then object is substituted.

```
// Source is unbounded
public class Node<T> {
    private T data;
    private Node<T> next;

    public Node(T data, Node<T> next) }
        this.data = data;
        this.next = next;
    }

    public T getData() { return data; }
}

// Erasure substitution with 'object'
public class Node {
    private Object data;
    private Node next;

    public Node(Object data, Node next) {
        this.data = data;
        this.next = next;
    }

    public Object getData() { return data; }
}
```

So too for the type erasure of generic methods.

When compiling a class or interface that extends a parameterized class or implements a parameterized interface, the compiler may need to create a synthetic method, called a bridge method. Don't be surprised if you see a bridge method if you fuck up your consuming code and a bridge method appears in the stack trace.

```
// Source
public class Node<T> {

    private T data;

    public Node(T data) { this.data = data; }

    public void setData(T data) {
        System.out.println("Node.setData");
        this.data = data;
    }
}

public class MyNode extends Node<Integer> {
    public MyNode(Integer data) { super(data); }

    public void setData(Integer data) {
        System.out.println("MyNode.setData");
        super.setData(data);
    }
}

// After Type erasure the follow variable substitutions occur
public class Node {

    private Object data;

    public Node(Object data) { this.data = data; }

    public void setData(Object data) {
        System.out.println("Node.setData");
        this.data = data;
    }
}

public class MyNode extends Node {

    public MyNode(Integer data) { super(data); }

    public void setData(Integer data) {
        System.out.println(Integer data);
        super.setData(data);
    }
}

// And after type erasure the following bridge method is created.
class MyNode extends Node {

    // Bridge method generated by the compiler
    //
    public void setData(Object data) {
        setData((Integer) data);
    }
    public void setData(Integer data) {
        System.out.println("MyNode.setData");
        super.setData(data);
    }

    // ...
}
```

Non-Reifiable Types

A reifiable type is a type whose type information is fully available at runtime. This entails primitives, non-generic types, raw types, and invocations of unbound wildcards.

Non-reifiable types are types where information has been removed at compile-time by type erasure. This entails invocations of generic types that are not defined as unbounded wildcards.

```
// List<String> and List<Number>
```

Heap pollution occurs when a variable of a parameterized type refers to an object that is not of that parameterized type.

Avoid heap pollution by compiling your code without warnings.

Heap pollution can be caused when defining a non-reifiable parameter to a varargs method.

```
public class ArrayBuilder {

  public static <T> void addToList (List<T> listArg, T... elements)
{
    for (T x : elements) {
      listArg.add(x);
    }
  }

  public static void faultyMethod(List<String>... l) {
    Object[] objectArray = l;     // Valid
    objectArray[0] = Arrays.asList(42);
    String s = l[0].get(0);       // ClassCastException thrown here
  }

}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/nonReifiableVarargsType.html

If you properly handle heap pollution can be caused when defining a non-reifiable parameter to a varargs method you can add the adding the following annotation to static and non-constructor method declarations: @SafeVargs.

Restrictions on Generics

You cannot instantiate generic types with primitive types.

```
// Don't do this: Primitive types int and char cause a compile-time error.
Pair<int, char> p = new Pair<>(8, 'a');
```

```
// Do this: non-primitive types
Pair<Integer, Character> p = new Pair<>(8, 'a');

// Note 8 and 'a" are autoboxed to non-primitive types. So the implicit call above is
Pair<Integer, Character> p = new Pair<>(Integer.valueOf(8), new Character('a'));
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#instantiate

You cannot create instances of type parameters.

```
public static <E> void append(List<E> list) {
    E elem = new E();  // compile-time error
    list.add(elem);
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#createObjects

As a workaround to the restriction of instantiating type parameters, you can create an object of a type parameter through reflection.

```
public static <E> void append(List<E> list, Class<E> cls) throws Exception {
    E elem = cls.newInstance();   // OK
    list.add(elem);
}

// You can invoke the append method as follows:
List<String> ls = new ArrayList<>();
append(ls, String.class);
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#createObjects

You cannot declare static fields whose types are type parameters.

```
// Don't do this. Compile error
public class MobileDevice<T> {
    private static T os;
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#createStatic

You cannot use instanceof with parameterised types.

```
public static <E> void rtti(List<E> list) {
    if (list instanceof ArrayList<Integer>) {  // compile-time error
        // ...
    }
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#cannotCast

You can, however, check that, in our example, the list is an ArrayList using unbounded wildcards.

```
public static void rtti(List<?> list) {
    if (list instanceof ArrayList<?>) {  // OK; instanceof requires a reifiable type
        // ...
    }
}

// when given ...
S = { ArrayList<Integer>, ArrayList<String>, LinkedList<Character>, ... }

// The runtime does not keep track of type parameters, so it cannot tell the difference
```

```
between an ArrayList<Integer> and an ArrayList<String>. However it does check if the
parameter is an ArrayList.
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#cannotCast

Casts ?

```
// Typically, you cannot cast to a parameterized type unless it is parameterized by
unbounded wildcards. For example:

List<Integer> li = new ArrayList<>();
List<Number>  ln = (List<Number>) li;  // compile-time error

// However, in some cases the compiler knows that a type parameter is always valid and
allows the cast. For example:

List<String> l1 = ...;
ArrayList<String> l2 = (ArrayList<String>)l1;  // OK
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#cannotCast

You cannot create an array of parameterized types.

```
List<Integer>[] arrayOfLists = new List<Integer>[2];  // compile-time errors
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#createArrays

You cannot create a Throw Object of a parameterized type.

```
// Extends Throwable indirectly
class MathException<T> extends Exception { /* ... */ }    // compile-time error

// Extends Throwable directly
class QueueFullException<T> extends Throwable { /* ... */ // compile-time error
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#cannotCatch

You cannot catch a parameterized type instance.

```
public static <T extends Exception, J> void execute(List<J> jobs) {
    try {
        for (J job : jobs)
            // ...
    } catch (T e) {   // compile-time error
        // ...
    }
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#cannotCatch

You can, however, use a type parameter in a throws clause.

```
class Parser<T extends Exception> {
    public void parse(File file) throws T {     // OK
        // ...
    }
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#cannotCatch

You cannot overload a method where the formal parameter types of each overload erase to the same raw type. ?

```
/// A class cannot have two overloaded methods that will have the same signature after type
erasure.

public class Example {
    public void print(Set<String> strSet) { }
    public void print(Set<Integer> intSet) { }
}

// The overloads would all share the same classfile representation and will generate a
compile-time error.
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html#cannotOverload

Packages

Basics

Packages are namespaces.

A package is a grouping of related types. A package allows: a logical grouping; avoiding naming conflicts; and controlling access.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/package/packages.html

Note "types" means classes, interfaces, enumerations, and annotations.

To include a type in a package put a package statement at the top of the source file. It must be the first line in the source file. The can only be one package statement in a source file and it applies to all types in the file.

```
//in the Draggable.java file
package graphics;
public interface Draggable {
    . . .
}

//in the Graphic.java file
package graphics;
public abstract class Graphic {
    . . .
}

//in the Circle.java file
package graphics;
public class Circle extends Graphic
    implements Draggable {
    . . .
}
```

Although you don't have to have a package statement in a source file it is recommended.

Package naming convention:

- All lower case.
- Organisations use their reversed internet domain name to begin their package.

```
au.com.softmake.mypagckage;
```

Using Packages

Use a public package member (types comprising a package are "package members") by either:

- Referencing its fully qualified name;
- Importing the package member (good for referencing a few members from a package);
- Importing the member's entire package (good for referencing many members from a package);

```
// Fully qualified name
graphics.Rectangle myRect = new graphics.Rectangle();
```

```
// Import a package member
import grahics.Rectangle;

Rectangle myRect = new Rectangle(); // Using the imported member.

// Import an entire package
import graphics.*;

Circle myCircle = new Circle();
Rectangle myRectangle = new Rectangle();
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/package/usepkgs.html

You can also import static final fields (constants) and static methods from one or two classes.

```
// Consider that java.lang.Math class has
public static final double PI = 3.141592653589793;
public static double cos(double a)
{
    ...
}

// Usual reference is to prefix the class name
double r = Math.cos(Math.PI * theta);

// Import static final field
import static java.lang.Math.PI;

// Import all static final fields and static methods from package
import static java.lang.Math.*;

// Now you can reference the static final field and static method as follows
double r = cos(PI * theta);
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/package/usepkgs.html

Only use static imports when this makes code more readable. Generally static imports make reading code *less* readable.

`import` statements go at the beginning of the file after the package statement.

```
package helloworld.demo;

import grahics.Rectangle;
import java.util.ArrayList;
```

You can use a package member by referencing its simple name if:

- the code you are writing is in the same package as that member; or
- that member has been imported (individually or as part of a package).

The Asterisk

The asterisk is never used as a wildcard to import a subset of types in a package.

```
// Compile error
import graphics.A*;
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/java/package/usepkgs.html

The asterisk can also be used, in addition to specifying an entire package, the public nested classes of an enclosing class.

```
// Suppose the graphics.Rectangle class nests Rectangle.DoubleWide and Rectangle.Square.

// Import the Rectangle class and its nested classes
import graphics.Rectangle;
import graphics.Rectangle.*;

// The second import statement will not import the Rectangle class.
```

Java implicitly imports two packages for each source file: the java.lang package; and the current package (the package for the current file).

Java packages are conceptually hierarchical but not as a matter of referencing them. You must import each package individually.

```
// For example, the Java API includes a java.awt package, a java.awt.color package, a
java.awt.font package, and many others that begin with java.awt. . However, the
java.awt.color package, the java.awt.font package, and other java.awt.xxxx packages are not
included in the java.awt package

// imports all of the types in the java.awt package,
// but it does not import java.awt.color, java.awt.font, or any other java.awt.xxxx packages
import java.awt.*;

// If you want to use java.awt.color as well as those in java.awt then you
// import each package individually.
import java.awt.*;
import java.awt.color.*;
```

Name ambiguities and Static Import

If a member in one package shares its name with a member in another package and both packages are imported you must refer to each member by its fully qualified name (and therefore consider not importing the packages in the first place).

```
// Consider the Rectangle class in the package java.awt and the hypothetical graphics
package.

// Import packages that share Rectangle class
import java.awt.*;
import graphics.*;

// OK
graphics.Rectangle gRect;
java.awt.Rectangle jaRect;

// Compile error
Rectangle Rect;
```

Exceptions

Overview

For a conceptual, language independent, overview of error handling and exceptions see "Error Handling Concepts.docx":

\\Atlas\Documents\Sda\Info\Every Application\KB\Techniques\Error Handling\Error Handling Concepts.docx

An exception is an event which disrupts the normal flow of the program's instructions.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html

Any code can throw an exception: your code, code from a package written by someone else such as the packages that come with the Java platform, or the Java runtime environment. Regardless of what throws the exception, it's always thrown with the throw statement.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html

When an exception is thrown it looks for an *exception handler* in the current method and if it can't find one it goes up the call stack until one is found. If no exception handler is found the program terminates at the line that originally threw the exception.

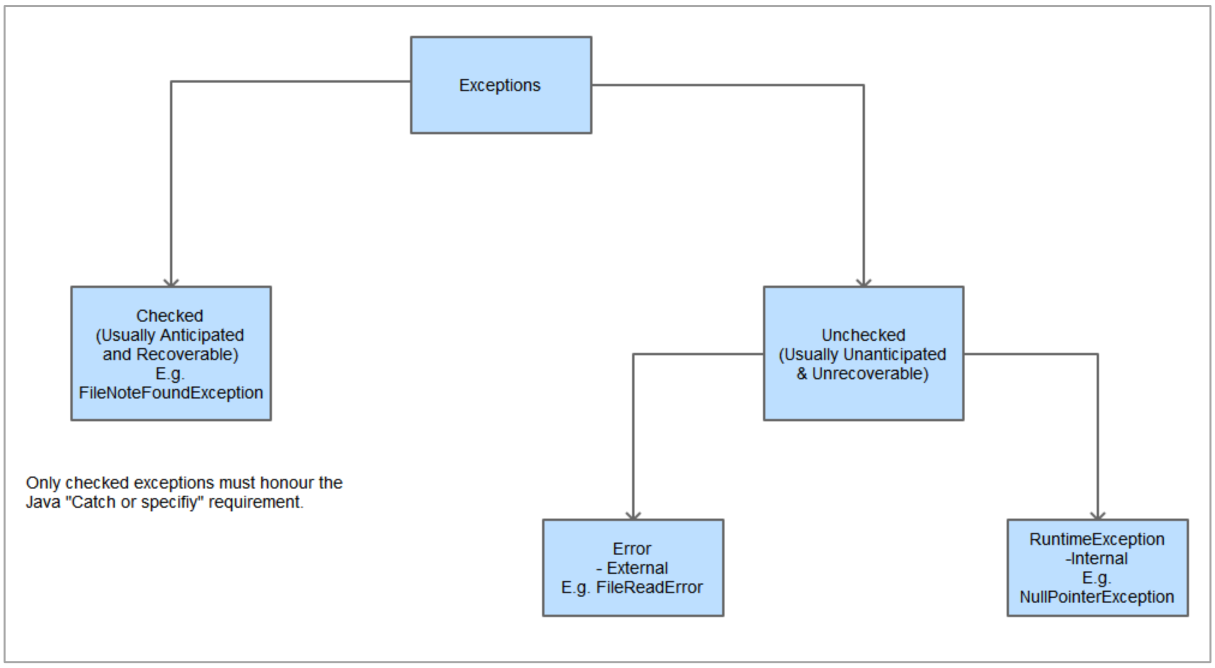
(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html

The three types of exceptions:

- Checked exceptions. Exceptional conditions that should be anticipated and recovered from. E.g. FileNotFoundException upon user supplies nonexistent file name.
- Errors. Exceptional conditions external to the application. The app usually cannot anticipate and recover from. E.g. IOError upon app being unable to read the file due to hardware or system malfunction.
- Runtime exceptions. Exceptional conditions internal to the application. The app usually cannot anticipate and recover from. These usually indicate programming bugs, such as logic errors or improper use of an API. E.g. NullPointerException upon null being passed to a constructor. It is better to eliminate the bug rather than catch and recover from these exceptions.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/catchOrDeclare.html

The types of Java exceptions are divided into *checked or unchecked exceptions* as follows:



Checked exceptions, and only checked exceptions, are subject to the *Catch or specify requirement.*

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/essential/exceptions/catchOrDeclare.html*

The *Catch or Specify requirement* entails that code that might throw a checked exception must be enclosed by either by: a try statement that also provides an exception handler for the exception; or a method that specifies that the exception can be thrown.

```
// OK as exceptionPlay() can throw a checked exception but is surrounded by a try statement
// with an exception handler
static public void start() {
  System.out.println("ExceptionsDemo");

  try {
    exceptionPlay();
  } catch (IOException e) {
    e.printStackTrace();
  }
}

// OK as checked exception is specified.
static private void exceptionPlay() throws IOException {
  // Checked exception.
  throw new IOException();
}
```

If code violates the *Catch or Specify requirement* it will not compile.

Try, catch, finally

Exception Reference Example showing try, catch, and finally clause.

```
static private void writeList() {
  PrintWriter out = null;
  final int SIZE = 10;

  // create an empty Vector vector with an initial capacity of SIZE
  Vector<String> vector = new Vector<String>(SIZE);
  vector.add("Sydney");
  vector.add("Melbourne");
  vector.add("Brisbane");

  // Possible checked exceptions are enclosed in a try statement with an exception handler.
  // So the "Catch or Specify" requirement is satisfied.
  try {

    System.out.println("Entering try statement");

    out = new PrintWriter(new FileWriter("OutFile.txt"));
    for (int i = 0; i < SIZE; i++)
      out.println("Value at: " + i + " = " + vector.elementAt(i));

  } catch (ArrayIndexOutOfBoundsException e) { // Unchecked, runtime exception.
    // E.g. we tried to access a vector element that doesn't exist.

    System.err.println("Caught ArrayIndexOutOfBoundsException: " + e.getMessage());

  } catch (IOException e) { // Checked Exception.
    // E.g. the program can't write to the file because it is locked.

    System.err.println("Caught IOException: " + e.getMessage());

  } finally { // Runs regardless of whether an exception was caught or not.
    // May not run if program terminates abnormally (e.g. power outage).

    if (out != null) {
      System.out.println("Closing PrintWriter");
      out.close();
    } else {
      System.out.println("PrintWriter not open");
    }
  }
} // writeList
```

For code that might throw an exception you can:

- Surround each line in a try block and provide and exception handlers for each; or
- Put all the code in a single try block and provide multiple handlers (generally recommended, as in the example above);
- Let another method higher in the call stack handle the exception (in which case you'll have to specify the exception in the current method for checked exceptions);
- Let the program halt execution at the offending line;

Catch clauses will match any ancestor class of the exception thrown.

```
// Matches ArrayIndexOutOfBoundsException, a descendent of Exception
} catch (Exception e) {
  System.err.println("Caught Exception: " + e.getMessage());
```

The runtime system checks a methods catch clauses (the exception handlers) in the order in which they appear after the try statement.

(Oracle, 2012) *http://docs.oracle.com/javase/tutorial/essential/exceptions/putItTogether.html*

With multiple catch clauses order the exceptions from most specific to least specific. Otherwise a compile error will occur: "Unreachable catch block for X. It is already handled by the catch block for Y".

```
// Don't do this ...
} catch (Exception e) {
  System.err.println("Caught Exception: " + e.getMessage());

// Unreachable catch block for ArrayIndexOutOfBoundsException. It is already handled by the
// catch block for Exception ...
} catch (ArrayIndexOutOfBoundsException e) { // Unchecked, runtime exception.
  // E.g. we tried to access a vector element that doesn't exist.

  System.err.println("Caught ArrayIndexOutOfBoundsException: " + e.getMessage());

}
```

Catching an exception allows the program to continue execution (unless you terminate the program in the catch clause).

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/catch.html

Java SE 7: A single catch block can handle more than one type of exception (but only if an exception is not a subclass of the other(s) in the list).

```
catch (IOException | SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/catch.html

The finally block always executes, whether an exception was caught or not (unless there has been an abnormal termination).

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/finally.html

The finally block is useful for more than just exception handling — it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

Clean up code in the finally block helps you avoid code duplication. It is a key tool for preventing resource leaks (closing files, connections, etc).

In general you want to catch the most specific exception as possible.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/advantages.html

Javs SE7: Use the try-with-resources statement for executing code containing resources. A resource is an object that must be closed after the program is finished with it (e.g. a file, database connection, etc). try-with-resources operate on objects that implement java.lang.AutoCloseable (this includes all objects that implement java.io.Closable).

```
// br will be automatically closed whether the try statement exits normally or as a result
// of an exception being thrown.
static String readFirstLineFromFile(String path) throws IOException {
    try (BufferedReader br =
                   new BufferedReader(new FileReader(path))) {
        return br.readLine();
    }
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Java SE7: You can declare several resources with a try-with-resources statement. The close methods of resources are called in the opposite order of their creation.

```
try (
    java.util.zip.ZipFile zf =
         new java.util.zip.ZipFile(zipFileName);
    java.io.BufferedWriter writer =
        java.nio.file.Files.newBufferedWriter(outputFilePath, charset)
) {
    // Enumerate each entry
    for (java.util.Enumeration entries =
                            zf.entries(); entries.hasMoreElements();) {
        // Get the entry name and write it to the output file
        String newLine = System.getProperty("line.separator");
        String zipEntryName =
             ((java.util.zip.ZipEntry)entries.nextElement()).getName() +
             newLine;
        writer.write(zipEntryName, 0, zipEntryName.length());
    }
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Java SE7: In a try-with-resources statement, any catch or finally block is run after the resources declared have been closed.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Specify Exceptions

Specify an exception when you want a method further up the call stack to handle it, rather than the current method.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/putItTogether.html

You can *Specify* multiple exceptions.

```
public void writeList() throws IOException, ArrayIndexOutOfBoundsException {
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/putItTogether.html

Throwing Exceptions

All exceptions are descendants of the Throwable class.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html

Throw an exception:

```
throw new EmptyStackException();
```

You can throw a built in or custom exception.

In general don't throw a RuntimeException.

Chaining Exceptions

You can *chain exceptions* by throwing an exception in response to another exception.

```
try {
} catch (IOException e) {
    throw new SampleException("Other IOException", e);
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/chained.html

Useful Throwable methods for chaining exceptions.

```
Throwable getCause()
Throwable initCause(Throwable)
Throwable(String, Throwable)
Throwable(Throwable)
```

Custom Exceptions

Define

```
public class MyCustomException extends Exception {
  public MyCustomException() {
    // TODO Auto-generated constructor stub
  }

  public MyCustomException(String message) {
    super(message);
    // TODO Auto-generated constructor stub
  }

  public MyCustomException(Throwable cause) {
    super(cause);
    // TODO Auto-generated constructor stub
  }

  public MyCustomException(String message, Throwable cause) {
    super(message, cause);
    // TODO Auto-generated constructor stub
  }
}
```

Use

```
// Throw the custom exception.
static private void exceptionPlay02() throws MyCustomException {
  // Checked exception.
  throw new MyCustomException();
}

// Call and handle code that might trigger the custom exception.
try {
  exceptionPlay02();
} catch (MyCustomException e) {
  e.printStackTrace();
}
```

Don't be tempted to avoid the *Catch or Specify requirement* by using unchecked exceptions in place of checked exceptions.

Why did the designers decide to force a method to specify all uncaught checked exceptions that can be thrown within its scope? Any Exception that can be thrown by a method is part of the method's public programming interface. Those who call a method must know about the exceptions that a method can throw so that they can decide what to do about them. These exceptions are as much a part of that method's programming interface as its parameters and return value.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html

If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception.

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html

Custom Stack Trace and Logging

Custom Stack Trace information (but note you could just use e.printStackTrace()).

```
} catch (Exception e) {
  // java.lang.StackTraceElement
  StackTraceElement stackTraceElements[] = e.getStackTrace();

  for(int i = 0, n = stackTraceElements.length; i < n; i++) {
    System.err.println(stackTraceElements[i].getFileName()
        + ":" + stackTraceElements[i].getLineNumber()
        + ">> " + stackTraceElements[i].getMethodName() + "()");
  }
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/chained.html

Custom Logging (logs as xml)

```
private static void errorLogger(Exception e) throws SecurityException, IOException {
  final String logFileName = "OutFile.log";
  FileHandler handler = new FileHandler(logFileName);
  Logger.getLogger("").addHandler(handler);

  Logger logger = Logger.getLogger("helloworld");
  StackTraceElement stackTraceElements[] = e.getStackTrace();
  for (int i = 0, n = stackTraceElements.length; i < n; i++) {
    logger.log(Level.WARNING, stackTraceElements[i].getFileName() + ":"
        + stackTraceElements[i].getLineNumber() + ">> " +
stackTraceElements[i].getMethodName()
        + "()"
        + e.getClass().getCanonicalName());
  }
  System.err.println("Errors logged to: " + logFileName);
}
```

Advantages of Exceptions

Exceptions, compared to the traditional branch and recover, allow the code to be simplified.

```
// Traditional branch and recover
errorCodeType readFile {
    initialize errorCode = 0;

    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;
        } else {
            errorCode = errorCode and -4;
        }
    } else {
        errorCode = -5;
    }
    return errorCode;
}

// Exceptions make it simpler
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
       doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

(Oracle, 2012) http://docs.oracle.com/javase/tutorial/essential/exceptions/advantages.html

You can propagate exceptions up the call stack.

You can group and categorize exception types.

Bibliography

(2018). Retrieved from Stackoverflow: https://stackoverflow.com/
Bentley, J. (2004, Aug 12). EcmaScript in Web Quick Reference.
Bentley, J. (2013, Aug). Java Reference - Framework.docx.
Bentley, J. (2013, Jul 05). Java Reference - Language - Printf Style Formatting.docx. Retrieved from "C:\Users\John\Documents\Sda\Info\Java\KB\Reference\Java Reference - Printf Style Formatting.docx"
Bentley, J. (2013, Jul). Java Reference - Language - Regular Expressions.docx.
Bentley, J. (2013, Jul). TutorialAtOracle Code Examples - HelloWorld. Retrieved from C:\Users\John\Documents\Sda\Code\Java\Examples\TutorialAtOracle\HelloWorld
Bentley, J. (2013, Jul). TutorialAtOracle Code Examples - Regex. Retrieved from C:\Users\John\Documents\Sda\Code\Java\Examples\TutorialAtOracle\Regex
Katz, D. (2012, Aug 22). *Avoiding 'Source not found' when I debug in Eclipse.* Retrieved Jul 01, 2013, from Jayway: http://www.jayway.com/2012/08/22/avoiding-source-not-found-when-i-debug-in-eclipse/
Mansoor, U. (2013, Aug 25). *Unicode isn't harmful for health – Unicode Myths debunked and encodings demystified.* Retrieved Aug 27, 2013, from 10K-LOC: http://10kloc.wordpress.com/2013/08/25/plain-text-doesnt-exist-unicode-and-encodings-demystified/
Oracle. (2011). *Java Platform, Standard Edition 6, API Specification*. Retrieved Jul 09, 2013, from http://docs.oracle.com: http://docs.oracle.com/javase/6/docs/api/overview-summary.html
Oracle. (2012). *The Java Tutorials.* Retrieved Jun 14, 2012, from http://docs.oracle.com/: http://docs.oracle.com/javase/tutorial/
ploygenelubricants. (2010, Aug 15). *The difference between a bounded type parameter and a upper bounded wildcard.* Retrieved from Stackoverflow: http://stackoverflow.com/a/3486711/872154
thenewboston. (2010, Feb 28). *Intermediate Java Tutorial - 17 - Generic Methods* . Retrieved Jun 02, 2013, from Java (Intermediate) Tutorials Youtube Playlist: http://www.youtube.com/watch?v=J6B_qauxfuc
Wikipedia. (n.d.). *List of Unicode Characters.* Retrieved from Wikipedia: http://en.wikipedia.org/wiki/List_of_Unicode_characters

Document Licence