# Java and Spring Boot for C#/.NET Programmers
# -
# Fundamentals

**Student Workbook**

Version 2023.02.05

**Presented to**
**The Harford Insurance Group**
**6 February, 2023**

**Author**

Paul Kimball
Interface Associates

# Table of Contents

# Module 1

# Welcome to Java!

# Section 1-1

# Course Overview

# Our Learning Goals

- **Read and write programs with Java and its core libraries**
  - Find out where to get downloads and documentation
  - Explore Java language syntax
  - Explore the most useful standard libraries

- **Use the Eclipse IDE**
  - Take advantage of the editor's features for writing good code
  - Debug, test and profile code before (or after) deployment
  - Work with external build tools and repositories

- **Use the maven build engine to manage dependencies and produce properly packaged code**

- **Understand the Spring Framework libraries**
  - Spring framework
  - Spring MVC
  - Spring Web
  - Spring Data
  - Spring Boot

- **Demystify the "magic" that Spring Boot performs**

- **Leverage your knowledge of C# and .NET concepts and apply them to Java**

# Our Approach

- **Focus on Java SE 1.8.0_351**
  - Newer Java release features may be mentioned without elaboration
- **Short, focused lectures with relatively simple examples**
  - Easy to read, but not always most elegant
- **Many links to online references and source code**
  - Use them, and Google frequently
- **Short labs that illustrate typical usage**
  - Code-along, to show language and IDE features
  - Self-propelled to develop your skills
- **Generally, a bottom-up approach to the language and tools**
  - Core language and JVM concepts in order so they build on each other
  - Demystify the "magic" of high-level frameworks like Spring Boot
- **Ask questions!**
  - If you don't ask me, I'll ask you...

# About Me

- **Paul Kimball**
  - Email:
    - pekimball@interfaceassociates.net
  - Github:
    - https://github.com/pek-ia
  - Linkedin:
    - https://www.linkedin.com/in/paul-e-kimball-interface-associates/

- **Cat Dad with 50 years of programming experience**
  - ... starting with hand-punched cards on a Wang 600

- **Significant experience in Education, Information Technology, and Aerospace industries**

- **Favorite languages include FORTRAN, C, C++, and Java**

- **Hobbies include cooking, music, photography, glassworking**

# Meet My Boss

- **His name is Bai-Bai, and he is pretty much the boss of me**

# Section 1-2

# Introducing Java

# TL;DR

- **One-stop shopping for info about Java SE 8 is at Oracle:**
  - **https://docs.oracle.com/javase/8/**
  - **https://docs.oracle.com/javase/tutorial/java/index.html**

- **Everything about Spring Boot is here:**
  - **https://spring.io/projects/spring-boot**

- **If you use maven, look no further:**
  - **https://maven.apache.org/**

- **If you prefer gradle, try this:**
  - **https://docs.gradle.org**

- **Tired of Visual Studio? Want to use Eclipse?**
  - https://www.eclipse.org/

- **For everything else, stick around**
  - Or Google it…

# "Write Once, Run Anywhere"

- **Java started the revolution in software portability**
  - Java 1.0 was developed by Sun Microsystems and released in 1996

- **The Java Virtual Machine (JVM) is a portable execution environment for diverse hardware/software platforms**
  - Linux, Windows, Mac, Android, Embedded systems, etc.
  - Similar to the .NET Common Language Runtime

- **The Java Runtime Environment (JRE) provides a large library of classes that support desktop, web, mobile and backend applications**
  - Some are essential to the language and are called the Platform API
  - Similar to .NET Base Class Library

- **The Java Software Development Kit (JDK) provides a curated set of tools and utilities for building Java applications**
  - including the Java compiler, debugger, and packaging tools

- **Java applications compile down to a common byte code format called "class files"**
  - These are portable between JVMs
  - Many other languages also target the JVM by producing class files
    - e.g., Kotlin, Scala, Clojure, Jython, jRuby - even Micro Focus COBOL!

# Java Specifications

- **Java has a *specification* that describes the language keywords, syntax, and semantics**
  - There are numerous implementations of Java compilers

- **The Java Virtual Machine has a *specification* for how class files are loaded, linked and executed at run time**
  - There are numerous implementations of the JVM on different operating systems and hardware platforms

- **The Java Platform API *specifies* core library classes that are used by many portions of the JDK and user apps**
  - e.g., java.lang.Object, the base class for all Java classes

- **To read the Java Language and JVM specifications, look here:**
  - **https://docs.oracle.com/javase/specs/**

- **To read the Java SE 8 Platform API documentation, look here:**
  - **https://docs.oracle.com/javase/8/**
  - You will spend a lot of time here

# Free and Open Source

- **Java specifications are overseen by the Java Community Process**
  - Defines and manages Java Specification Requests (JSRs) to add new features to the standards
  - When someone talks about "JSR 330" (or something like that) look here:
    - **https://jcp.org**

- **Development is done by members of the OpenJDK Community**
  - They work on the code that implements JSRs
    - Under the Oracle Contributor Agreement
  - They build the JDK tools according to JDK Enhancement Proposals (JEPs)
  - When people say "Project Valhalla" or "Project Loom", or "JEP 161" look here:
    - **https://openjdk.org**

- **The *reference implementations* of the JVM, JRE and JDK are open source code on github**
  - If you want to read source code and know how it *really* works, look here:
    - https://github.com/openjdk/

- **Oracle acquired the trademark and Sun Microsystems in 2010**
  - Oracle drives the release schedule
  - Extended support from Oracle will cost you

- **There are plenty of proprietary implementations of Java compilers, JVMs, and libraries**
  - There are many Products and Frameworks layered on top of Java

# Java Downloads

- **Java SE 8 distributions can be downloaded from Oracle:**
  - https://www.oracle.com/java/technologies/javase/javase8u211-later-archive-downloads.html

- **Java SE Runtime Environment (JRE)**
  - The "public" JRE
  - Provides the JVM and platform libraries so you can run programs that have been compiled to class files
  - Does not include any development tools

- **The Java SE Development Kit (JDK)**
  - Provides the compiler, linker, debugger, jar management tools, performance monitors, etc.
  - Does not include any standard Java editor or workbench

- **Documentation can be downloaded separately**
  - https://www.oracle.com/java/technologies/javase-jdk8-doc-downloads.html
  - In case you want to work offline

# Free Java IDEs

- **Interactive Development Environments are easy to get**
  - All of them support additional languages and features through plugins
    - e.g., PHP, HTML5, CSS, JavaScript, C, C++, Kotlin, Scala
    - Even non-C languages like FORTRAN and COBOL
  - Most are Open Source
  - Here are some of the most popular:

- **Eclipse IDE**
  - Download and explore at the Eclipse Foundation
    - https://www.eclipse.org
    - Open source
    - The base for many derivative products, e.g., Spring Tool Suite (STS), MyEclipse

- **Apache NetBeans**
  - Download and explore at the Apache Software Foundation
    - https://netbeans.apache.org/
    - Open source

- **IntelliJ IDEA**
  - Download and explore at JetBrains
    - https://www.jetbrains.com/idea/
    - Community Edition is open source
    - Derivative products include Android Studio

- **Visual Studio Code**
  - Download and explore at Microsoft
    - https://code.visualstudio.com/
    - A handy and good looking editor

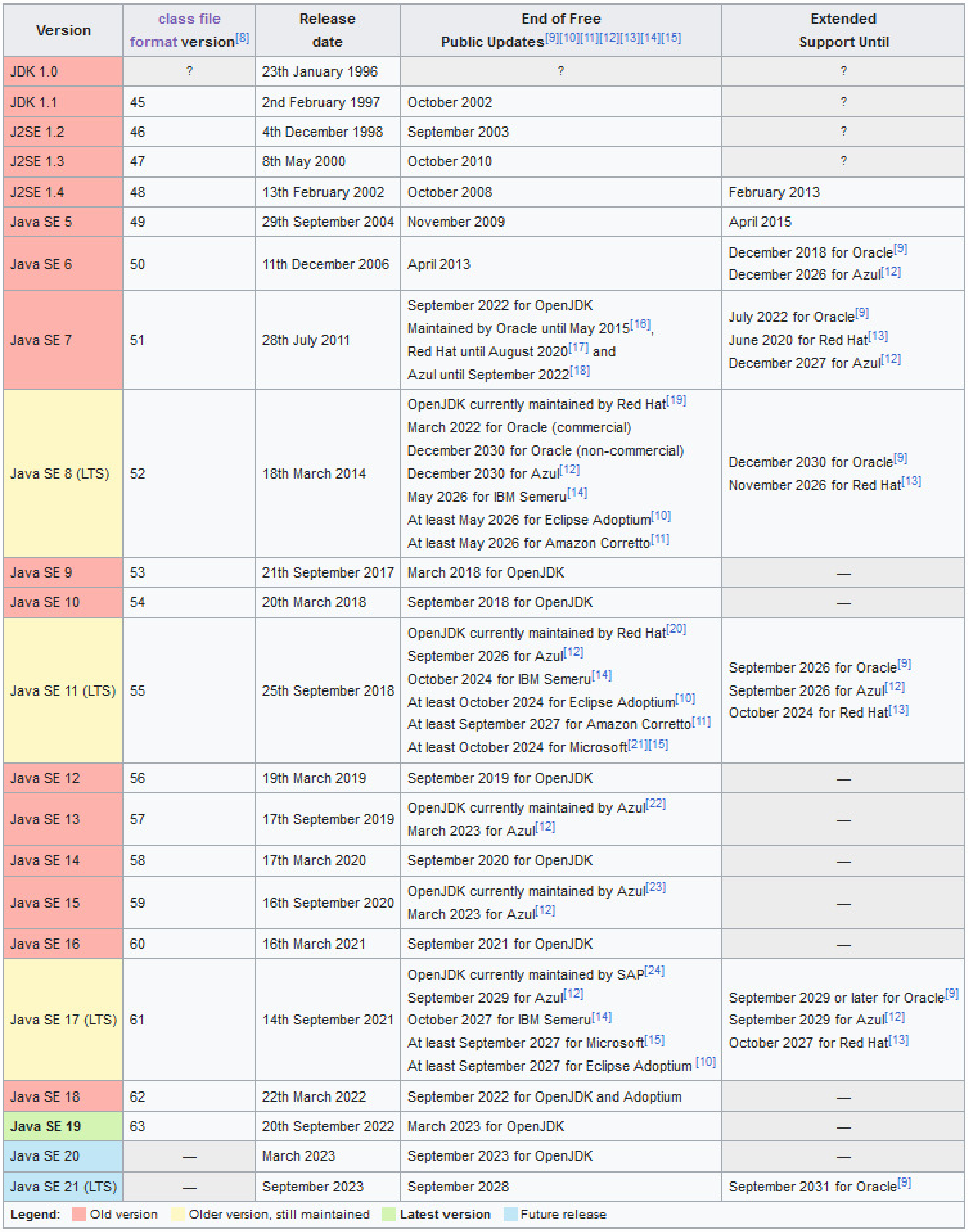
- **Many, many others, including JDeveloper from Oracle, extensions to emacs, vim**

# Java Language Features

- **A general-purpose, object-oriented programming language**
  - You can build just about anything
- **Like C#, Java is a member of the C/C++ language family**
  - Classes, interfaces, methods, member variables, loops and logic will look familiar to any C# programmer
- **Emphasizes portability, familiarity, and simplicity**
  - Deliberately hides complicated features of C and C++ programs
    - Hardware native types
    - Pointers and addresses
    - Memory management
    - Preprocessor and #include files
    - Operator overloading
- **A dynamic language**
  - Class discovery, loading, resource optimization and configuration is done at runtime, rather than compile time
  - HotSpot JVM optimizes on the fly
  - Objects carry extensive run-time type information for reflection
- **Automatic garbage collection**
  - Unused objects are automatically removed from memory
  - Garbage collector can be tuned for different environments
- **A ton of immediately useful libraries**
  - Threads, concurrency, networking, I/O, collections, reflection, etc.
  - Most of Java programming is learning about the libraries
- **Base platform for many important technology stacks**
  - Apache tomcat, Spring Boot, Hibernate, Java EE, Minecraft, etc.

# Version History

See Wikipedia: [Java Version History]



| Version | class file format version[8] | Release date | End of Free Public Updates[9][10][11][12][13][14][15] | Extended Support Until |
|---|---|---|---|---|
| JDK 1.0 | ? | 23th January 1996 | ? | ? |
| JDK 1.1 | 45 | 2nd February 1997 | October 2002 | ? |
| J2SE 1.2 | 46 | 4th December 1998 | September 2003 | ? |
| J2SE 1.3 | 47 | 8th May 2000 | October 2010 | ? |
| J2SE 1.4 | 48 | 13th February 2002 | October 2008 | February 2013 |
| Java SE 5 | 49 | 29th September 2004 | November 2009 | April 2015 |
| Java SE 6 | 50 | 11th December 2006 | April 2013 | December 2018 for Oracle[9]<br>December 2026 for Azul[12] |
| Java SE 7 | 51 | 28th July 2011 | September 2022 for OpenJDK<br>Maintained by Oracle until May 2015[16], Red Hat until August 2020[17] and Azul until September 2022[18] | July 2022 for Oracle[9]<br>June 2020 for Red Hat[13]<br>December 2027 for Azul[12] |
| Java SE 8 (LTS) | 52 | 18th March 2014 | OpenJDK currently maintained by Red Hat[19]<br>March 2022 for Oracle (commercial)<br>December 2030 for Oracle (non-commercial)<br>December 2030 for Azul[12]<br>May 2026 for IBM Semeru[14]<br>At least May 2026 for Eclipse Adoptium[10]<br>At least May 2026 for Amazon Corretto[11] | December 2030 for Oracle[9]<br>November 2026 for Red Hat[13] |
| Java SE 9 | 53 | 21th September 2017 | March 2018 for OpenJDK | — |
| Java SE 10 | 54 | 20th March 2018 | September 2018 for OpenJDK | — |
| Java SE 11 (LTS) | 55 | 25th September 2018 | OpenJDK currently maintained by Red Hat[20]<br>September 2026 for Azul[12]<br>October 2024 for IBM Semeru[14]<br>At least October 2024 for Eclipse Adoptium[10]<br>At least September 2027 for Amazon Corretto[11]<br>At least October 2024 for Microsoft[21][15] | September 2026 for Oracle[9]<br>September 2026 for Azul[12]<br>October 2024 for Red Hat[13] |
| Java SE 12 | 56 | 19th March 2019 | September 2019 for OpenJDK | — |
| Java SE 13 | 57 | 17th September 2019 | OpenJDK currently maintained by Azul[22]<br>March 2023 for Azul[12] | — |
| Java SE 14 | 58 | 17th March 2020 | September 2020 for OpenJDK | — |
| Java SE 15 | 59 | 16th September 2020 | OpenJDK currently maintained by Azul[23]<br>March 2023 for Azul[12] | — |
| Java SE 16 | 60 | 16th March 2021 | September 2021 for OpenJDK | — |
| Java SE 17 (LTS) | 61 | 14th September 2021 | OpenJDK currently maintained by SAP[24]<br>September 2029 for Azul[12]<br>October 2027 for IBM Semeru[14]<br>At least September 2027 for Microsoft[15]<br>At least September 2027 for Eclipse Adoptium [10] | September 2029 or later for Oracle[9]<br>September 2029 for Azul[12]<br>October 2027 for Red Hat[13] |
| Java SE 18 | 62 | 22th March 2022 | September 2022 for OpenJDK and Adoptium | — |
| **Java SE 19** | 63 | 20th September 2022 | March 2023 for OpenJDK | — |
| Java SE 20 | — | March 2023 | September 2023 for OpenJDK | — |
| Java SE 21 (LTS) | — | September 2023 | September 2028 | September 2031 for Oracle[9] |

**Legend:** 🟥 Old version 🟨 Older version, still maintained 🟩 **Latest version** 🟦 Future release

# Version Compatibility

- **Generally, older class files can run on newer JVMs**
  - The Java SE 19 JVM supports class file versions 46+
- **The compiler can produce older class file representations**
  - The Java SE 19 compiler can produce class files compatible with Java SE 7 and later JVMs
  - Controlled by --target option
- **The compiler can enforce older source code syntax**
  - The Java SE 19 compiler can enforce source code compliance back as far as Java SE 7
  - Controlled by --source option
- **The target release must be equal to or higher than the source release**

# Java Evolution

- **The Java language architects know that there are many implementations of Compilers and JVMs, and have their priorities**
  - Don't break backwards compatibility
    - Don't add an operator or a keyword if a library would do
    - Don't expose hardware details
    - Consider changes to the JVM VERY VERY carefully
    - OpenJDK curates a giant set of test cases
  - Change is slow... JREs and Projects can go on for years and years

- **The Platform API developers know that libraries are big, and have their own priorities**
  - Don't expand the Platform library unless everyone needs it
  - Move towards proper modularization of the Platform APIs
    - Support tiny devices
    - Hide internal APIs
    - Update public APIs to match real world needs
  - Change is slow... e.g., Compact profiles in Java SE 8, modules in Java SE 9, tooling still in work

# Java Revolution

- **Everyone else just wants to code, but they all have different apps and priorities**
  - Change must be fast!  If Java language or Platform API doesn't do it, write something new!
  - Introduce features as fast as possible via new frameworks!
  - Explore all that can be done with Reflection, Class Loaders, Annotations and native methods (written in C)

- **Cool features show up first as libraries or tools, and may (*eventually*) get folded back into Platform library, language, or even the JVM**

- **There are multiple public, competing, open-source libraries for any particular abstraction**
  - Graphics, Web page template engines, Web services, Data Acesss

- **We don't get one standard library, we get twenty!**

# Comparing Java and C#

- **Little differences - we'll cover them today**
  - Basic data types
  - Capitalization conventions
  - Language syntax
  - Classes and Interfaces

- **Ordinary differences we'll cover this week**
  - Data type hierarchy
  - APIs for important libraries like networking or I/O
  - OO encapsulation, inheritance and polymorphism
  - Threads
  - Lambdas

- **Perplexing differences will require a long time to absorb**
  - Build tools and project directory structures
  - Packaging and distribution of libraries
  - Repositories
  - Concepts exposed as keywords in C# vs libraries in Java
    - Example: await/yield keywords vs Future<> class
  - Dependencies

# Java Big Concepts

- **Portability / Hardware independence**
  - Explains some features
- **Organization of source and compiled code**
  - Source code and bytecode in "packages"
  - "modules" introduced in Java SE 9
  - Deployment in JAR files
- **Run-time configuration**
  - Class Loaders (and CLASSPATH)
  - Reflection
  - Annotations as a general extension mechanism
- **Native C methods as a necessary evil**
  - um, what happened to "portability"? ...

# Let's Try It!

- **Let's go old-school!**
- **Open a command window and follow along with me as we illustrate a VERY IMPORTANT point about Java by writing Hello World**

# Hello World

- **The source file `HelloWorld.java` can be typed into a simple text editor or created in an IDE**
  - Holds a single method named `main`, which prints out a message

**Example**

```
/* This is the file HelloWorld.java
   The class name is the same as the file name,
   but without the .java extension
*/

public class HelloWorld {

  // A Java main program
  public static void main( String[] args ) {
    System.out.println( "Welcome to Java!" );
  }
}
```

- **The class name is `HelloWorld`**

- **Java is case-sensitive, everywhere, always**
  - You will learn to depend on capitalization conventions
  - More on keywords and syntax later

# Compiling

- **The `javac` command runs the Java compiler**
  - Argument specifies the file(s) to be compiled
  - `.java` extension is required after source filenames

**Example**

```
prompt> javac HelloWorld.java
```

  - To compile all Java files in the current directory:

**Example**

```
prompt> javac *.java
```

- **The compiler creates the bytecode file `HelloWorld.class`**

# Running from the Command Line

- **The `java` command runs the Java virtual machine**
  - Argument specifies name of the *class* holding the main program

**Example**

```
prompt> java HelloWorld
Welcome to Java!
```

- **Just one problem, and it's a big one**
  - HelloWorld is in the default package, and we *never* want to do that

# Packages

- **A *package* is a group of related classes that reside in the same directory**
  - To assign a class to a package, declare the package name in the first non-comment line of the source file

**Example**

```
package com.pekia;

public class HelloWorld { ... }
```

- **The fully qualified class name is now `com.pekia.HelloWorld`**

- **The class file *must* go in a directory that matches its package name**
  - Each dot in the package name implies a subdirectory
  - `HelloWorld.class` *must* go in a directory named `com\pekia`
  - You can use the `–d` compiler option to create the appropriate subdirectories automatically

**Example**

```
prompt> javac –d . HelloWorld.java
```

- **Then use the *fully-qualified* class name when running**

**Example**

```
prompt> java com.pekia.HelloWorld
Welcome to Java!
```

- **C# programmers note:**
  - This is similar to the C# namespace

# JAR Files

- **Java applications use hundreds of `.class` files and directories**

- **For ease of handling, these are always packaged in JAR files**

- **A *Java ARchive* (JAR) file holds an entire directory tree**
  - PKZIP format
  - Filename usually has a `.jar` extension

- **C# programmers note:**
  - This is the equivalent of an assembly (.dll or .exe)

# The Class Loader

- **Java programs are linked at run time**

- **The *class loader* loads class files as they are needed in code**

- **The *default class loader* looks in the following places:**
  - Bootstrap classes in the JRE core JAR files
  - Installed extension classes in extension JAR files
  - **Finally, application-specific directories (or JARs) specified by the `CLASSPATH` environment variable or `java` command line**

# CLASSPATH

- **CLASSPATH IS ABSOLUTELY CRITICAL AT RUNTIME**

- **The classpath is a list of directories and/or JAR files**
  - Each entry is used, in order, as the root of a directory tree when resolving a fully-qualified class name
  - Default value is the current working directory `"."`

- **Let's say you compile like this:**

```
prompt> javac -d D:\MYWORK\bin HelloWorld.java
```

- **You can set the `CLASSPATH` environment variable**

```
prompt> SET CLASSPATH=%CLASSPATH%;.;D:\MYWORK\bin
prompt> java com.pekia.HelloWorld
```

- **... or pass as a command-line option to the JVM**

```
prompt> java –classpath D:\MYWORK\bin com.pekia.HelloWorld
```

  - This option overrides the `CLASSPATH` environment variable

- **You CANNOT set a classpath globally that satisfies all applications and drivers**
  - Most Java apps are run by scripts or executables that set `CLASSPATH` first, then run the JVM

# Importing Classes

- **Classes declared `public` can be used by code in other packages**
  - The imported class must be ON THE CLASSPATH at compile time and at run time
  - They can be referenced by their fully qualified names

```
java.awt.Button b = new java.awt.Button("Submit");
```

- ***Import* a class to make its simple name available**

```
// Imports go after package statement, but before class
import java.awt.Button;
...
// Short name is now OK
Button b = new Button("Submit");
```

- **Import an entire package with a wildcard `"*"`**

```
import java.awt.*;
```

  - but wildcard does *not* import classes in subdirectories

```
import java.awt.*;
import java.awt.event.*;  // must import explicitly
```

- **`import` has no effect on code size or execution time**
  - Just gives the compiler permission to look in specified packages to resolve short names

- **Classes in the package `java.lang` are always imported automatically**

- **Eclipse note:**
  - Eclipse will automatically search for simple class names in the project classpath, and will offer to write the import statement
  - Select Project menu -> Properties to select the Build Path and the JRE runtime

# Module 2

# Introducing the Eclipse IDE

# Base Software Setup

- Windows 10
- Java SE 8 (JDK + JRE)
- Eclipse JEE 2022.03 or later
- Maven
- Access to github.com public repositories
- On-Line References

# Free Java IDEs

- **General-purpose, customizable, cross-platform tools are easy to get**
  - All of them support additional languages and features through plugins
    - e.g., PHP, HTML5, CSS, JavaScript, C, C++, Kotlin, Scala
    - Even non-C languages like FORTRAN and COBOL
  - Most are Open Source
  - Here are some of the most popular:

- **Eclipse IDE**
  - Download and explore at the Eclipse Foundation
    - https://www.eclipse.org
    - Open source
    - The base for many derivative products, e.g., Spring Tool Suite (STS), MyEclipse

- **Apache NetBeans**
  - Download and explore at the Apache Software Foundation
    - https://netbeans.apache.org/
    - Open source

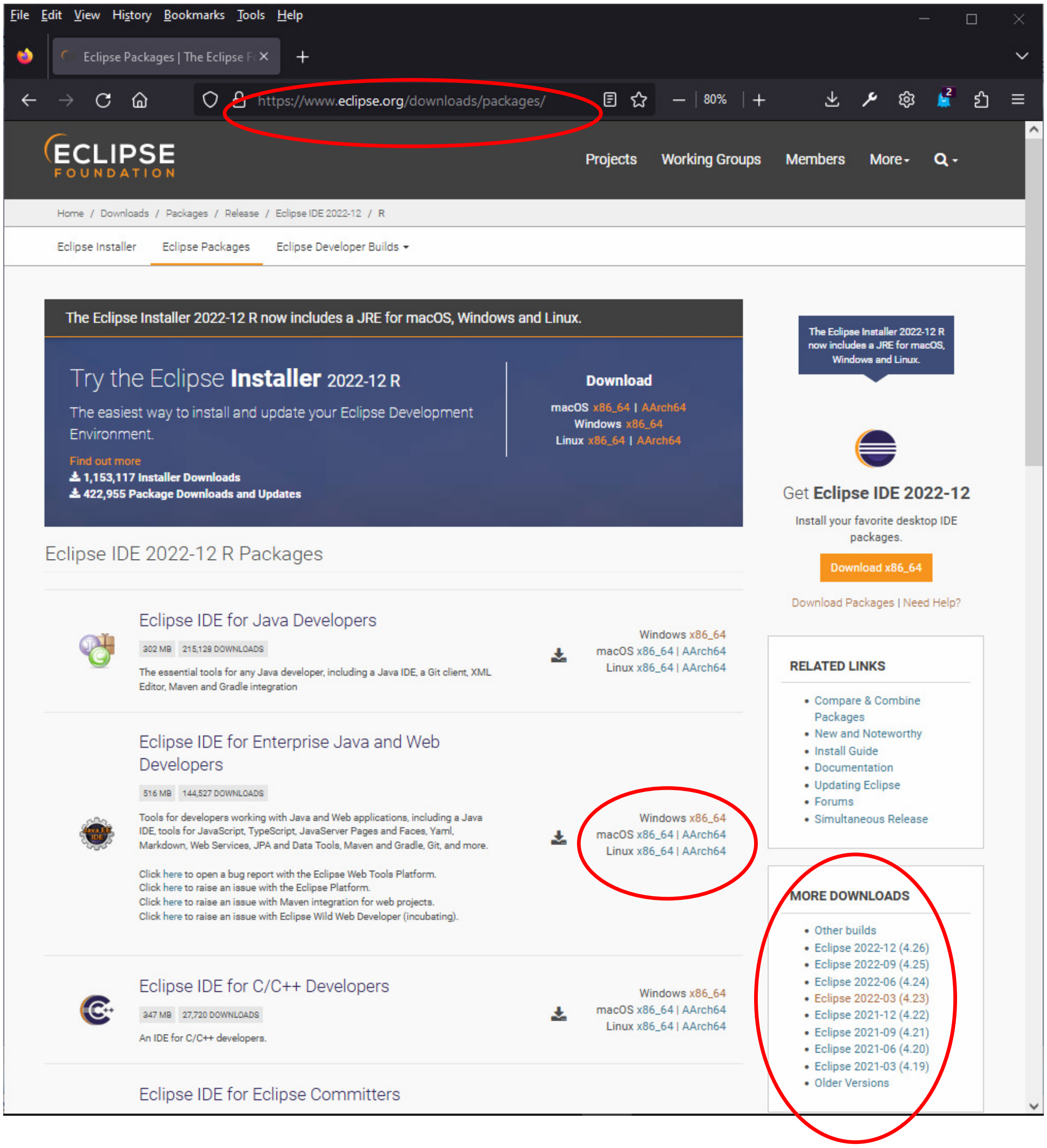
- **IntelliJ IDEA**
  - Download and explore at JetBrains
    - https://www.jetbrains.com/idea/
    - Community Edition is open source
    - Derivative products include Android Studio

- **Visual Studio Code**
  - Download and explore at Microsoft
    - https://code.visualstudio.com/
    - A handy and good looking editor

- **Many, many others, including JDeveloper from Oracle, extensions to emacs, vim**

# Install Eclipse

- **Browse installation packages on eclipse.org website**
  - o [https://www.eclipse.org/downloads/packages/](https://www.eclipse.org/downloads/packages/)


File Edit View History Bookmarks Tools Help
Eclipse Packages | The Eclipse F
https://www.eclipse.org/downloads/packages/
80%
ECLIPSE FOUNDATION
Projects
Working Groups
Members
More
Home / Downloads / Packages / Release / Eclipse IDE 2022-12 / R
Eclipse Installer
Eclipse Packages
Eclipse Developer Builds

The Eclipse Installer 2022-12 R now includes a JRE for macOS, Windows and Linux.

Try the Eclipse **Installer** 2022-12 R

The easiest way to install and update your Eclipse Development Environment.

Find out more

1,153,117 Installer Downloads

422,955 Package Downloads and Updates

**Download**

macOS x86_64 | AArch64
Windows x86_64
Linux x86_64 | AArch64

Eclipse IDE 2022-12 R Packages

Eclipse IDE for Java Developers

302 MB 215,128 DOWNLOADS

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration

Windows x86_64
macOS x86_64 | AArch64
Linux x86_64 | AArch64

Eclipse IDE for Enterprise Java and Web Developers

516 MB 144,527 DOWNLOADS

Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services, JPA and Data Tools, Maven and Gradle, Git, and more.

Click here to open a bug report with the Eclipse Web Tools Platform.
Click here to raise an issue with the Eclipse Platform.
Click here to raise an issue with Maven integration for web projects.
Click here to raise an issue with Eclipse Wild Web Developer (incubating).

Windows x86_64
macOS x86_64 | AArch64
Linux x86_64 | AArch64

Eclipse IDE for C/C++ Developers

347 MB 27,720 DOWNLOADS

An IDE for C/C++ developers.

Windows x86_64
macOS x86_64 | AArch64
Linux x86_64 | AArch64

Eclipse IDE for Eclipse Committers

The Eclipse Installer 2022-12 R now includes a JRE for macOS, Windows and Linux.

Get **Eclipse IDE 2022-12**

Install your favorite desktop IDE packages.

Download x86_64

Download Packages | Need Help?

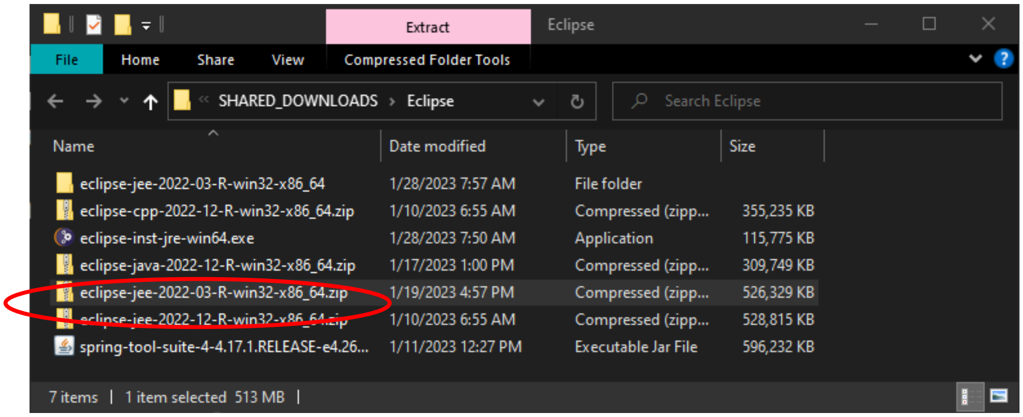
RELATED LINKS
- Compare & Combine Packages
- New and Noteworthy
- Install Guide
- Documentation
- Updating Eclipse
- Forums
- Simultaneous Release

MORE DOWNLOADS
- Other builds
- Eclipse 2022-12 (4.26)
- Eclipse 2022-09 (4.25)
- Eclipse 2022-06 (4.24)
- Eclipse 2022-03 (4.23)
- Eclipse 2021-12 (4.22)
- Eclipse 2021-09 (4.21)
- Eclipse 2021-06 (4.20)
- Eclipse 2021-03 (4.19)
- Older Versions

- **Download the version and the archive type (.zip, .dmg, .exe installer, .etc) that you want**

# Local installation (no privileges)

- **Download .zip archive from eclipse.org**

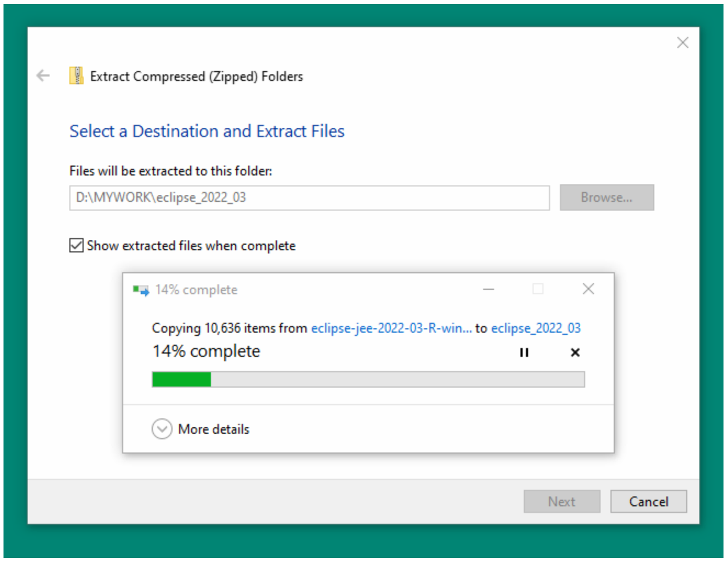


| Name | Date modified | Type | Size |
|---|---|---|---|
| eclipse-jee-2022-03-R-win32-x86_64 | 1/28/2023 7:57 AM | File folder | |
| eclipse-cpp-2022-12-R-win32-x86_64.zip | 1/10/2023 6:55 AM | Compressed (zipp... | 355,235 KB |
| eclipse-inst-jre-win64.exe | 1/28/2023 7:50 AM | Application | 115,775 KB |
| eclipse-java-2022-12-R-win32-x86_64.zip | 1/17/2023 1:00 PM | Compressed (zipp... | 309,749 KB |
| eclipse-jee-2022-03-R-win32-x86_64.zip | 1/19/2023 4:57 PM | Compressed (zipp... | 526,329 KB |
| eclipse-jee-2022-12-R-win32-x86_64.zip | 1/10/2023 6:55 AM | Compressed (zipp... | 528,815 KB |
| spring-tool-suite-4-4.17.1.RELEASE-e4.26... | 1/11/2023 12:27 PM | Executable Jar File | 596,232 KB |

7 items | 1 item selected 513 MB |

- **In your download directory, select the zip file; then right-click and select Extract All... from the context menu**

- **Extract to a folder that you can access**
  - in this example, D:\MYWORK\eclipse_2022_03



Extract Compressed (Zipped) Folders

Select a Destination and Extract Files

Files will be extracted to this folder:

D:\MYWORK\eclipse_2022_03

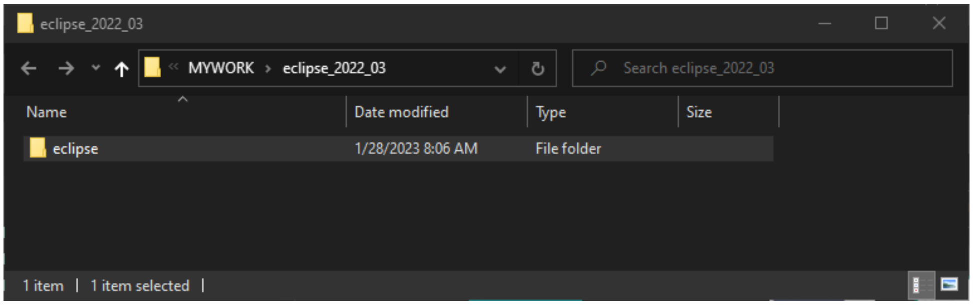
☑ Show extracted files when complete

14% complete

Copying 10,636 items from eclipse-jee-2022-03-R-win... to eclipse_2022_03

14% complete

- **Done; you have created the eclipse installation directory!**



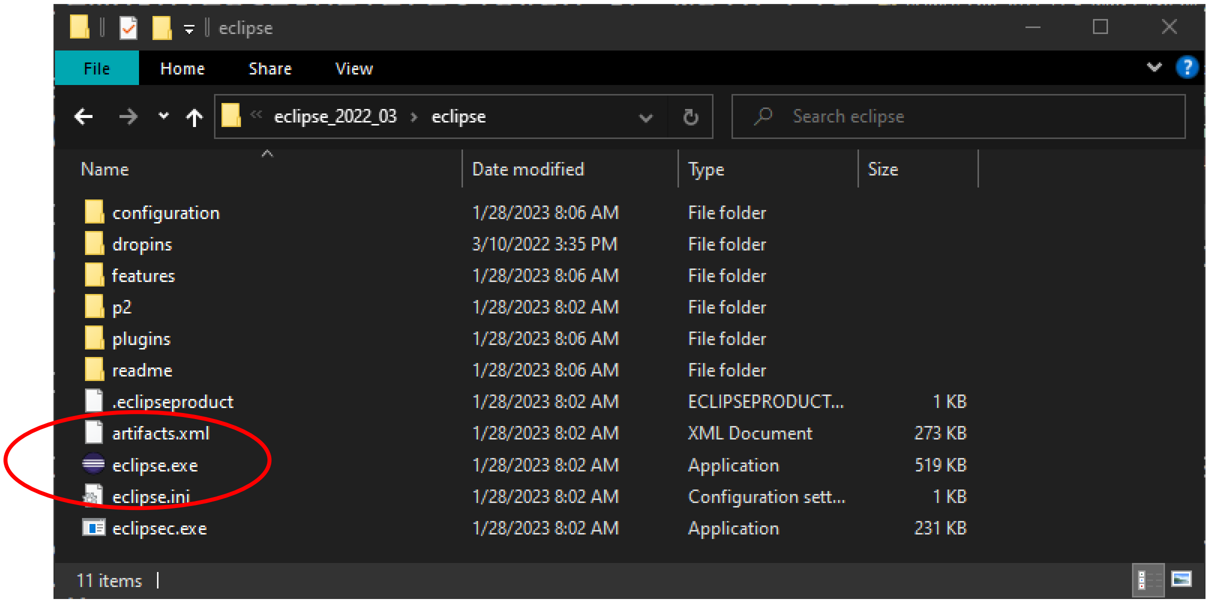
| Name | Date modified | Type | Size |
|---|---|---|---|
| eclipse | 1/28/2023 8:06 AM | File folder | |

1 item | 1 item selected |

# Follow Along With Me

- **Let's take it for a spin!**
- **Follow along as we:**
  - Start Eclipse
  - Select a workspace
  - Create a Java project
  - Write some code
  - Compile, run, and debug it
  - Customize the IDE a bit

# Start Eclipse

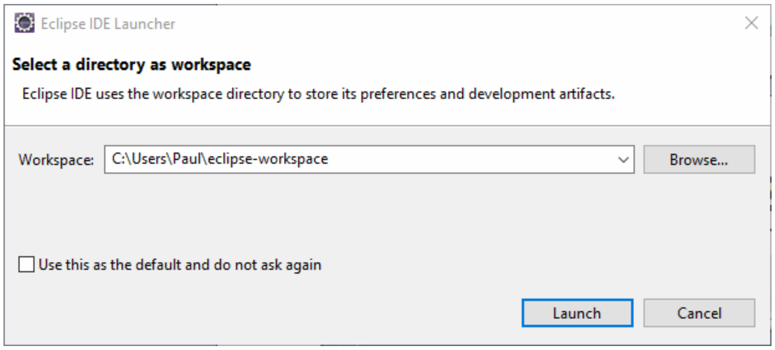


| Name | Date modified | Type | Size |
|---|---|---|---|
| configuration | 1/28/2023 8:06 AM | File folder | |
| dropins | 3/10/2022 3:35 PM | File folder | |
| features | 1/28/2023 8:06 AM | File folder | |
| p2 | 1/28/2023 8:02 AM | File folder | |
| plugins | 1/28/2023 8:06 AM | File folder | |
| readme | 1/28/2023 8:06 AM | File folder | |
| .eclipseproduct | 1/28/2023 8:02 AM | ECLIPSEPRODUCT... | 1 KB |
| artifacts.xml | 1/28/2023 8:02 AM | XML Document | 273 KB |
| eclipse.exe | 1/28/2023 8:02 AM | Application | 519 KB |
| eclipse.ini | 1/28/2023 8:02 AM | Configuration sett... | 1 KB |
| eclipsec.exe | 1/28/2023 8:02 AM | Application | 231 KB |

- **Run eclipse.exe**
  - For convenience, you can create a shortcut for eclipse.exe; then copy your desktop, pin to taskbar, pin to start
  - Add the installation directory to your %PATH% if you'd like to be able to start it from the command line
    - eclipsec.exe uses the command line as a logging console

- **The installation includes a built-in JRE, which runs the workbench**
  - Works out-of-the-box, even if you haven't installed another JDK or JRE

# Select a Workspace



Eclipse IDE Launcher

**Select a directory as workspace**

Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\Paul\eclipse-workspace

□ Use this as the default and do not ask again

- **A *workspace* is a directory where eclipse stores your work**
  - Think of it as your home base

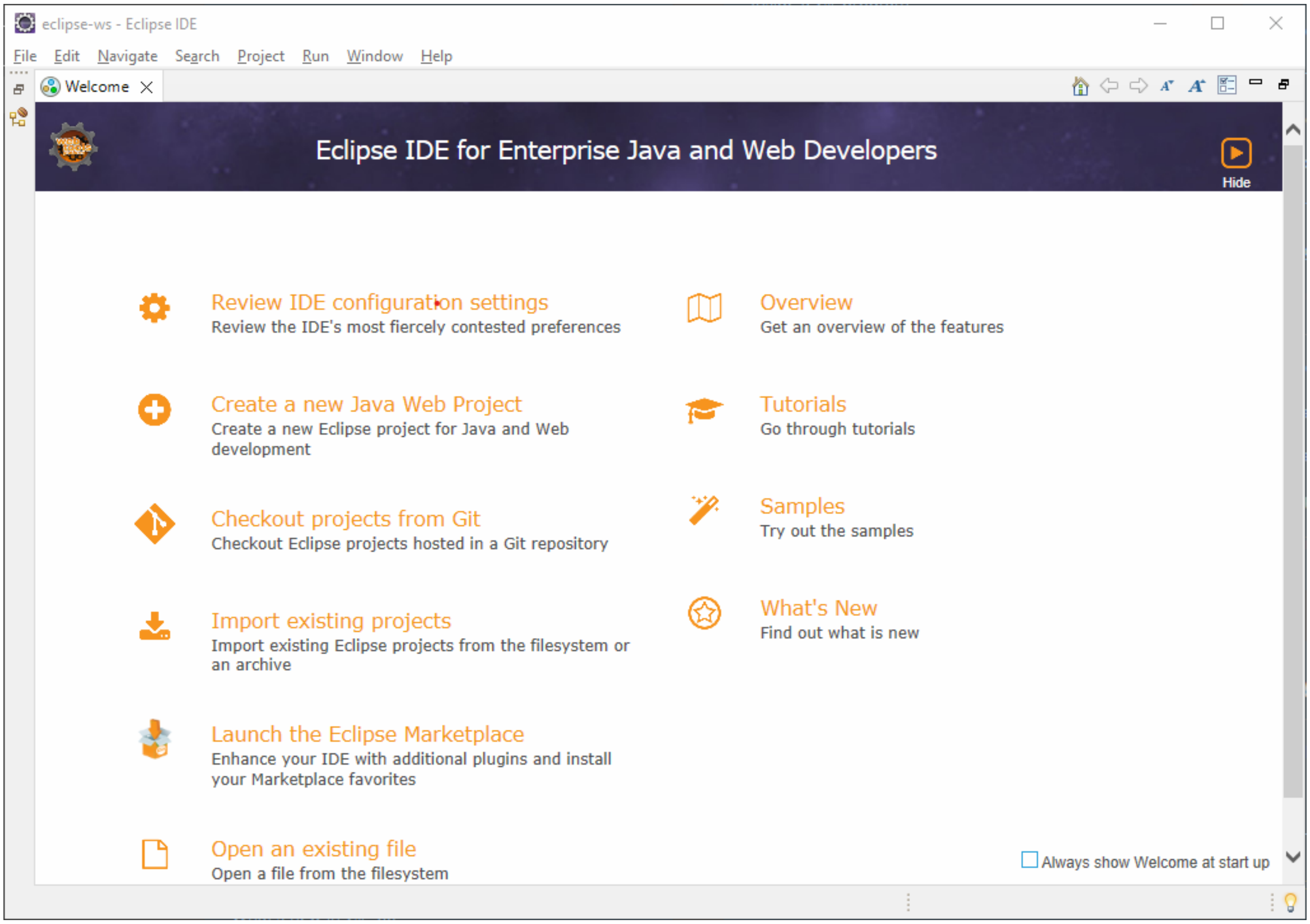
- **By default, eclipse creates a workspace in your home directory**
  - If you don't want this, change it to some other directory you can access

- **It doesn't matter what you name it, or where you put it, but please make sure you can find it easily during class**

- **You can have many different workspaces**
  - You might want to develop C++ in one, and build Java web apps in another
  - Each workspace has its own setup, look, tools and projects

# Welcome

- **The welcome screen is displayed the first time you run eclipse**


eclipse-ws - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Welcome
Hide
□ Always show Welcome at start up

Eclipse IDE for Enterprise Java and Web Developers

Review IDE configuration settings
Review the IDE's most fiercely contested preferences

Create a new Java Web Project
Create a new Eclipse project for Java and Web development

Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository

Import existing projects
Import existing Eclipse projects from the filesystem or an archive

Launch the Eclipse Marketplace
Enhance your IDE with additional plugins and install your Marketplace favorites

Open an existing file
Open a file from the filesystem

Overview
Get an overview of the features

Tutorials
Go through tutorials

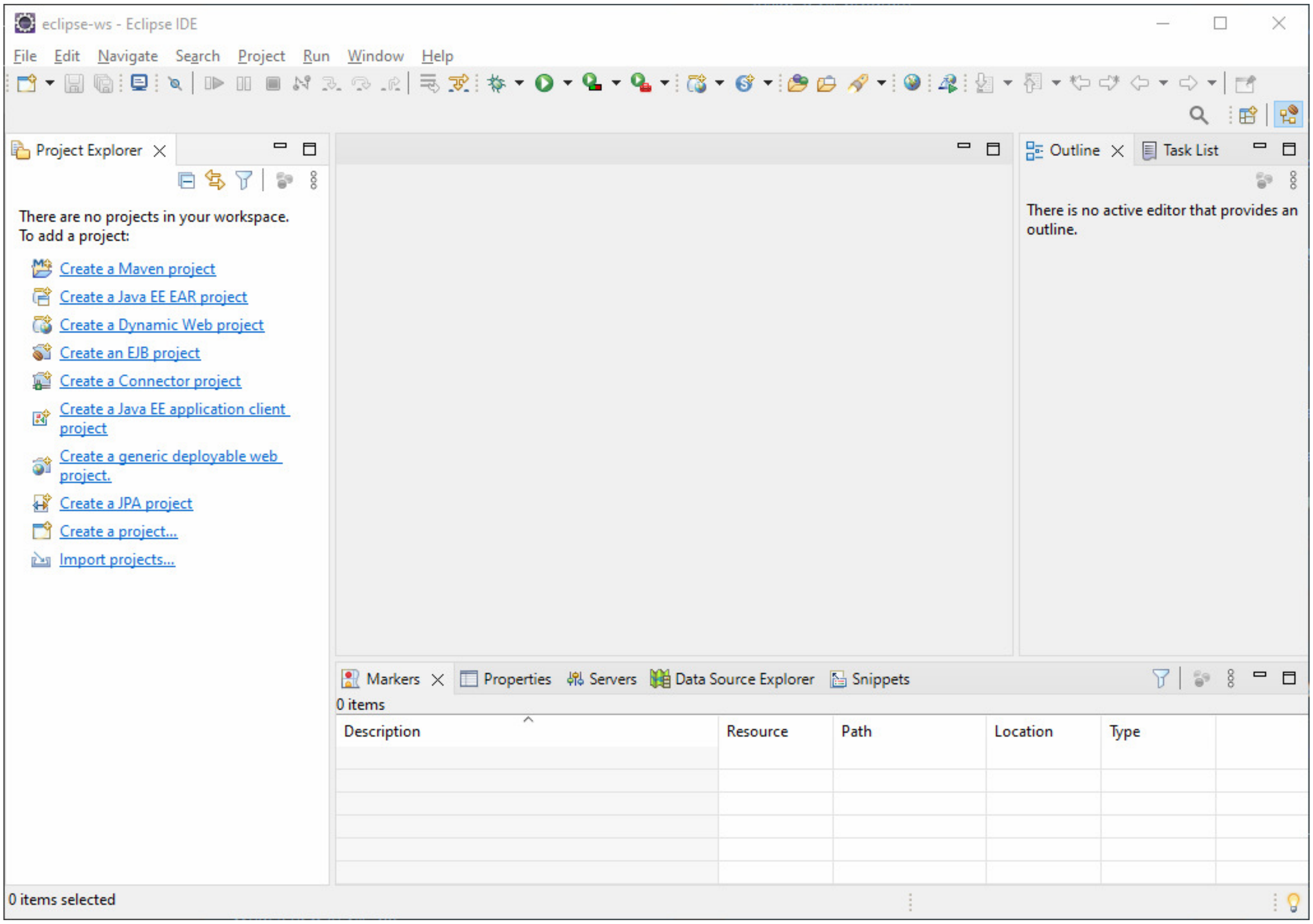
Samples
Try out the samples

What's New
Find out what is new

- **Dismiss it (for now) by clicking on the X in its tab**

# The Eclipse Workbench (empty)

- No projects yet



There are no projects in your workspace.
To add a project:

- Create a Maven project
- Create a Java EE EAR project
- Create a Dynamic Web project
- Create an EJB project
- Create a Connector project
- Create a Java EE application client project
- Create a generic deployable web project.
- Create a JPA project
- Create a project...
- Import projects...

There is no active editor that provides an outline.

0 items

| Description | Resource | Path | Location | Type |
|---|---|---|---|---|

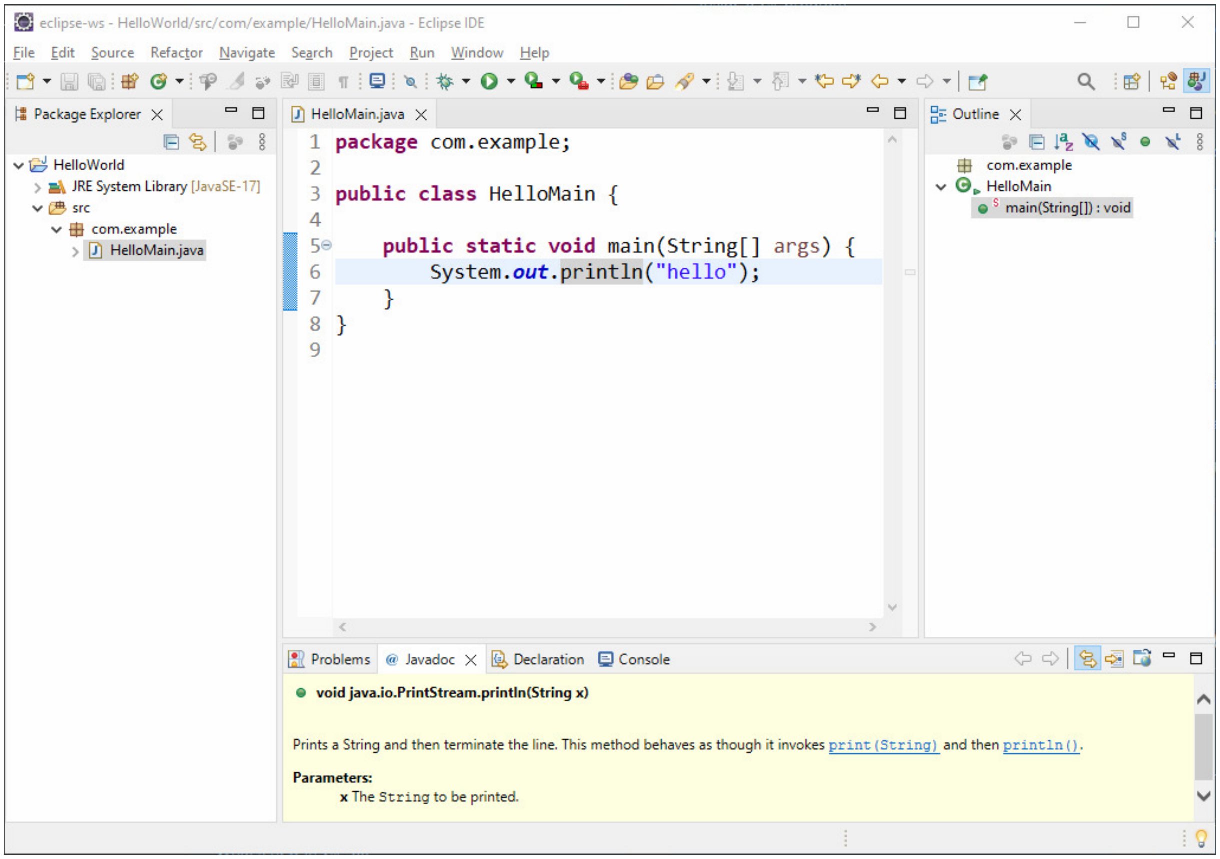
0 items selected

- **Old-school, big tool**
  - Menus with many choices
  - Toolbars with lots of icons
  - Tabbed, dockable panes to display different tools, views, files

- **As we go along we'll customize it and tidy it up**

# Eclipse Workbench

- **After building a project**



```java
package com.example;

public class HelloMain {

    public static void main(String[] args) {
        System.out.println("hello");
    }
}

```

● void java.io.PrintStream.println(String x)

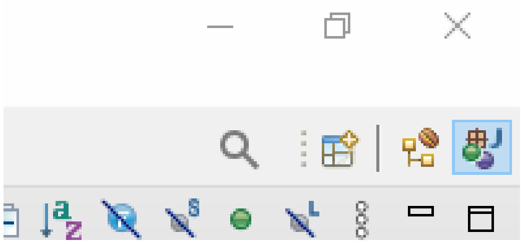
Prints a String and then terminate the line. This method behaves as though it invokes `print(String)` and then `println()`.

**Parameters:**
**x** The `String` to be printed.

# About the User Interface

- **Each tabbed window is called a *View***
  - To hide a View
    - Select the little X in the corner of its tab, or
    - Right-click in tab; select Close
  - To show a View
    - Select Window menu -> Show view -> *view name*
    - The view you want is often under Other...

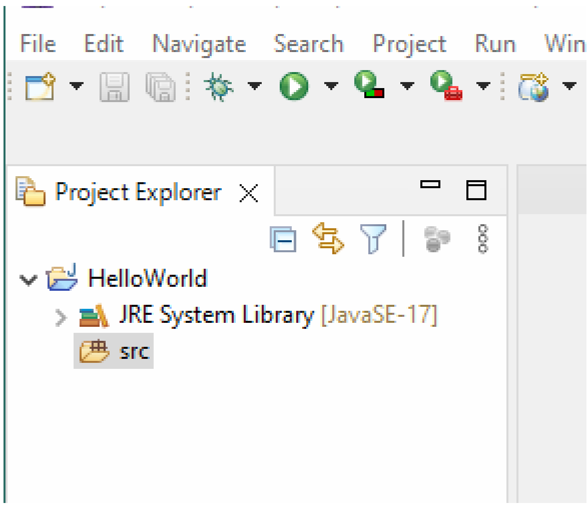
- **A prearranged set of views is called a *Perspective***
  - You'd probably use different views for debugging than you do for code management
  - To change your perspective
    - Select Window menu -> Open perspective -> *perspective name*
    - Or, way up in the upper right corner, click on one of the little perspective icons icons



# Create a Java Project

- **A *Project* is a set of resources (files, servers, etc.) that must be edited and maintained together**
  - Produces an artifact that can be deployed and managed
  - A JAR file, for a Java project

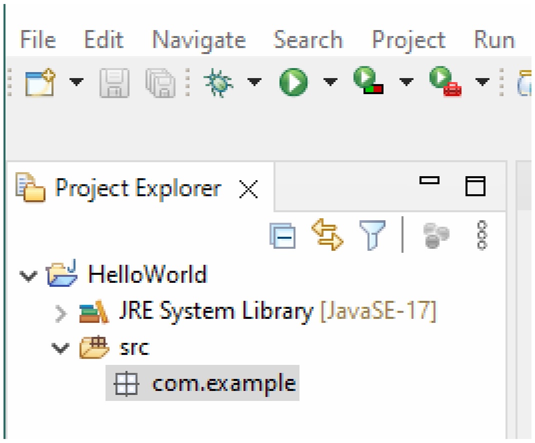
- **To create the Project**
  - Select File menu -> new -> Project...
  - Then, select "Java Project" from the list
  - In the Wizard, enter your project name, e.g. HelloWorld
  - Select the Finish button
  - You now have an empty project!



- **C# programmers note**
  - This is similar to how projects in VS produce assemblies

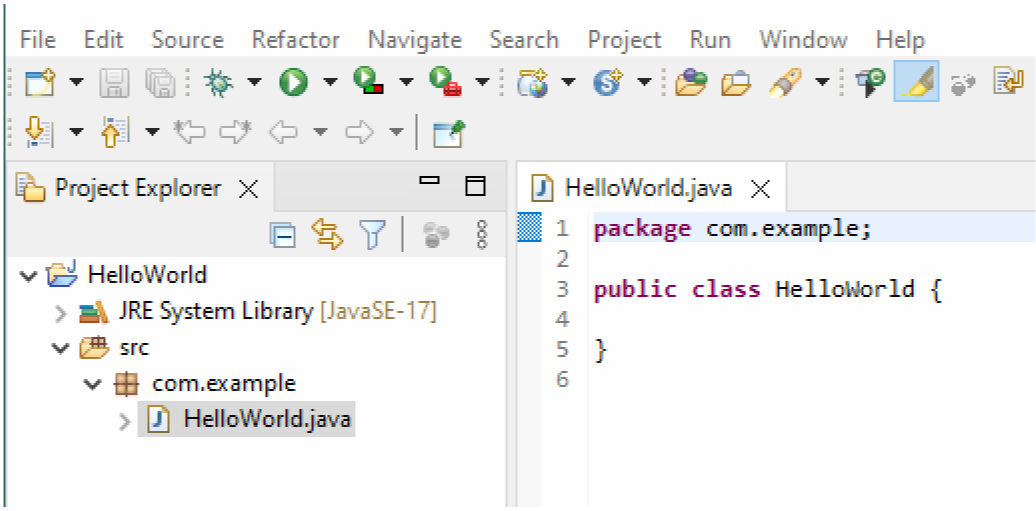
# Create a package

- **You don't want to use the "default" package!**
- **To create a package**
  - Select your src folder
  - Right-click to select context menu -> New -> Package
  - In the Java Package wizard, provide a name, e.g., `com.example`
    - All lower case, with dots for separators
  - Select Finish
  - You have an empty package directory!



# Create a Class

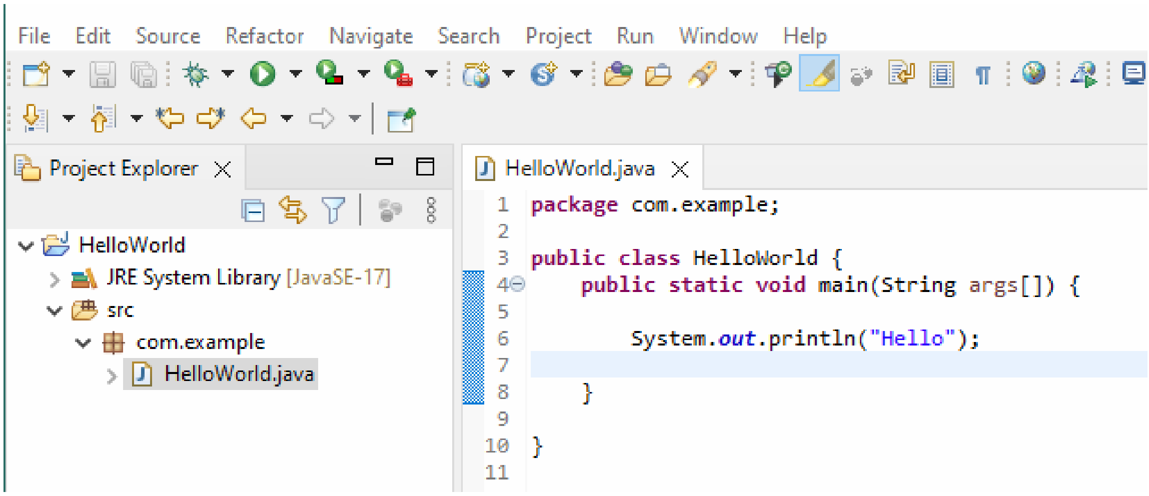
- **To create a Class**
  - Select your package folder
  - Right-click to select context menu -> New -> Class
  - In the Java Package wizard, provide a name, e.g., `HelloWorld`
    - One word, in PascalCase
  - Select Finish



```
package com.example;

public class HelloWorld {

}
```

# Write Some Code

- **Write a simple main program**
  - Ctrl-S to save it

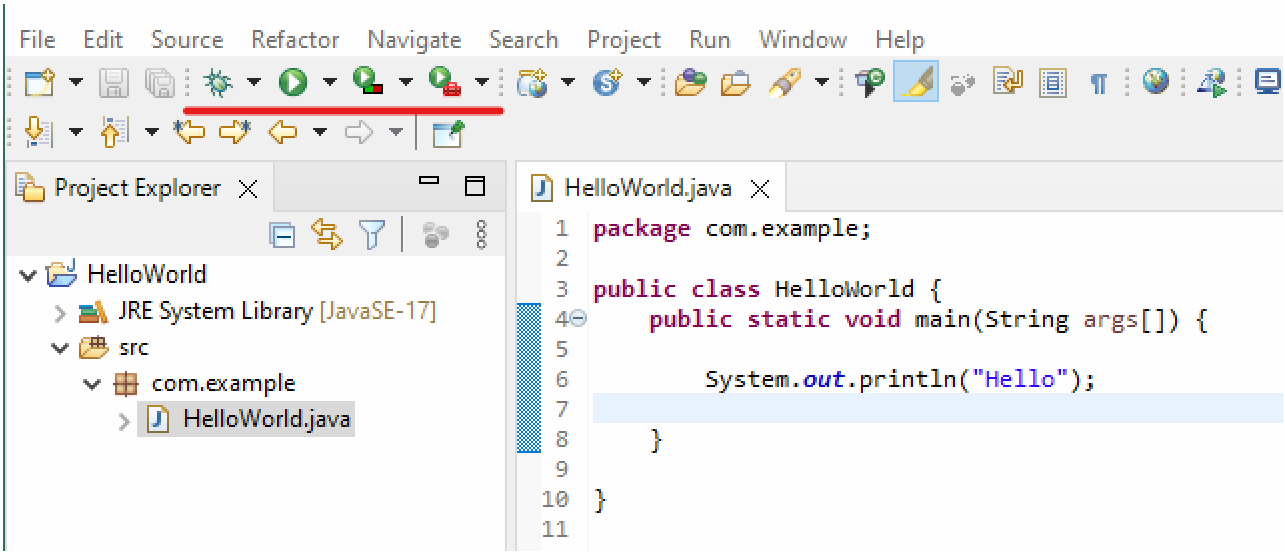


File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
HelloWorld
JRE System Library [JavaSE-17]
src
com.example
HelloWorld.java

```
package com.example;

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

        System.out.println("Hello");

    }

}
```

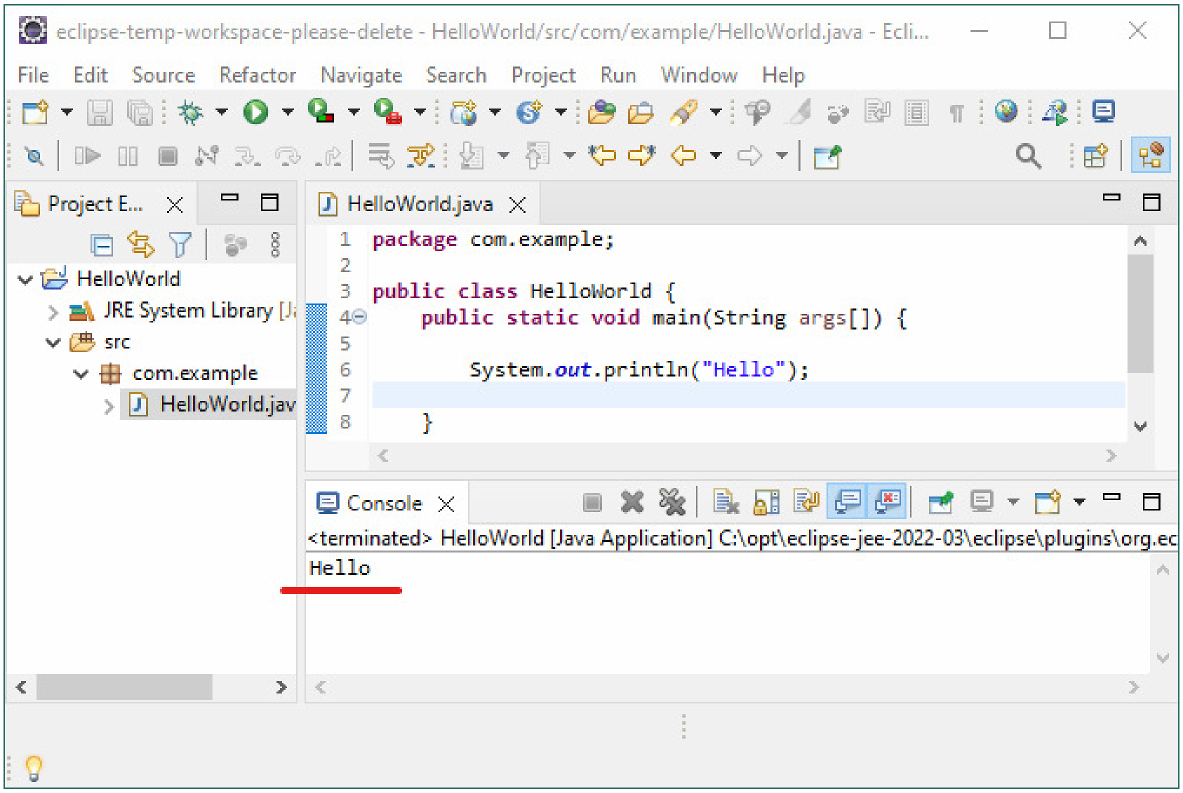
# Run It

- **Using the toolbar, select the green arrow button**



```
package com.example;

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

        System.out.println("Hello");

    }

}

```

- **It works!**



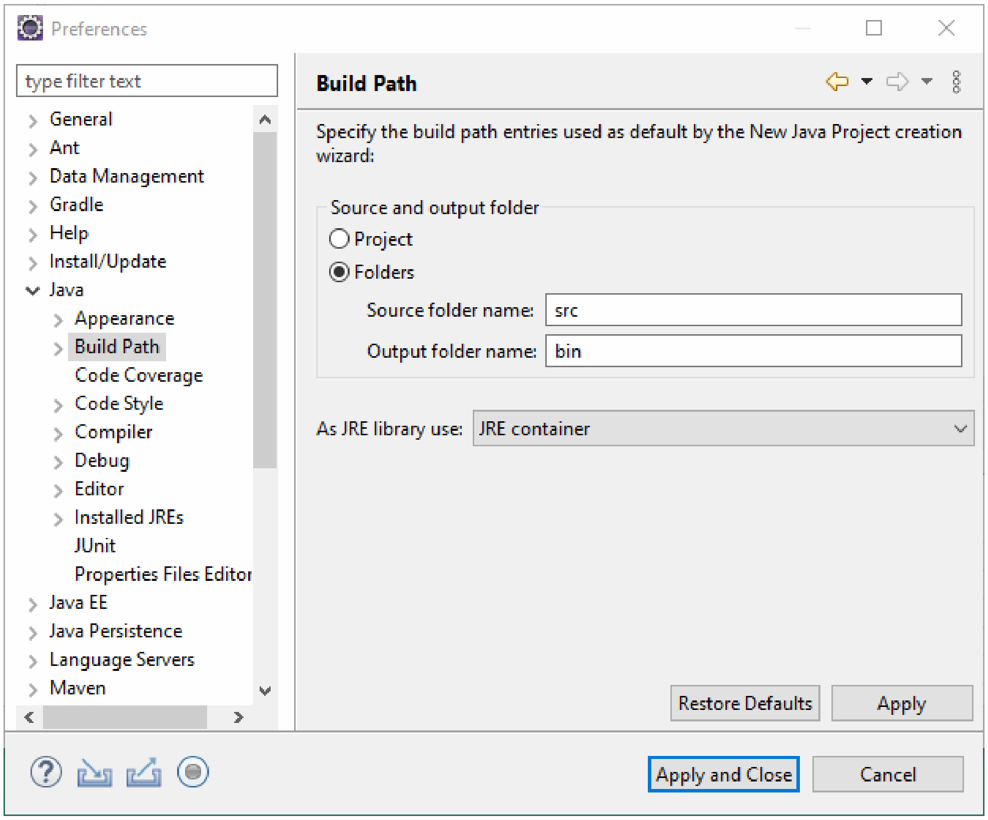
```
package com.example;

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

        System.out.println("Hello");

    }
```

```
<terminated> HelloWorld [Java Application] C:\opt\eclipse-jee-2022-03\eclipse\plugins\org.ec
Hello
```

# Workspace Preferences

- **These control the default behavior of Eclipse in this workspace**

- **Window menu -> Preferences**
  - And... BEHOLD ALL TEH OPTIONS !!!
  - Grouped in categories/subcategories
  - At least it has a "search" feature



**Build Path**

Specify the build path entries used as default by the New Java Project creation wizard:

Source and output folder

○ Project

● Folders

Source folder name: src

Output folder name: bin

As JRE library use: JRE container

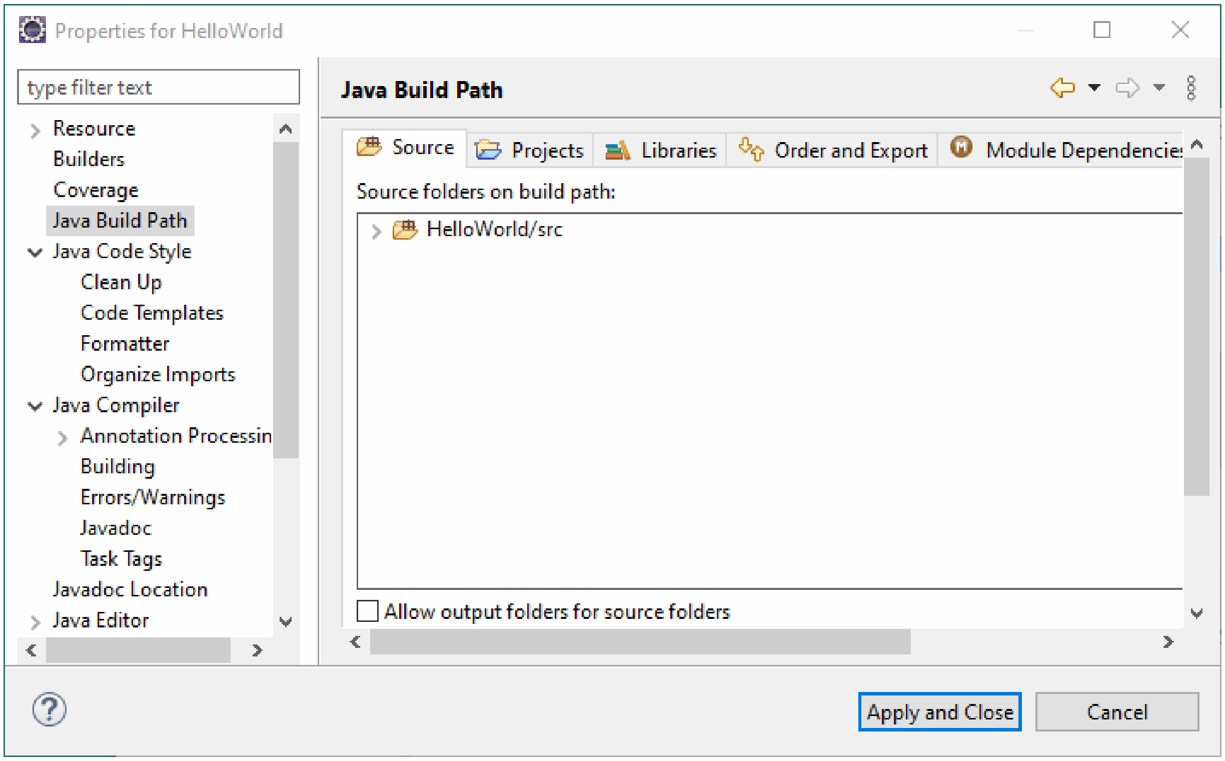
- **Make a cheatsheet for options you care about; google all else**
  - Some categories, at least, will become familiar to you

- **Workspace settings are stored in the .metadata hidden subdirectory**
  - List of current Projects, their types and status
  - Links to external repositories or directories
  - Eclipse config files
  - Current workbench settings and state
    - Eclipse can look very different in different workspaces!

# Project

- **Project Properties control how a particular project is managed, written, built, and executed**

- **Select Project menu -> Properties**



- **Some of these settings will override the workspace defaults**
  - e.g., Java Build Path, JRE for running the project, etc.

# Eclipse Plugins

- **Eclipse is the Sum of its Plugins**
  - Without plugins, it's an empty shell

- **Each plugin adds features to the workbench**
  - Views and Editors
  - Wizards to generate or manage code
  - Adapters to external systems, e.g. Git, Tomcat servers
  - New menu choices and items in settings dialogs

- **Pre-configured downloads are based on profiles**
  - Java Developer, Java for Enterprise Developer, C/C++ Developer, etc.
  - Profiles are defined by the Eclipse Packaging project
    - **https://projects.eclipse.org/projects/technology.packaging**

- **Ultimately, you can decide which plugins you keep**
  - Select Help menu -> About Eclipse IDE to manage which plugins are part of your installation

# Module 3

# Java Language Syntax

# Section 3-1

# Java Basics

# Twin Children of C / C++

- **The basic syntax of Java is the same as C#**
  - Case-sensitive identifiers
  - Comments are standard C-style block /* */ and end-of-line //
  - Each statement ends with a semicolon ( ; )
  - Whitespace is ignored unless part of a quoted string
  - Curly braces { } delimit code blocks, including class declarations, method bodies, compound statements, etc.
  - Expression syntax and operators are familiar (though not identical)
  - No global functions

- **Loops and logic are very similar**
  - conditional logic with if/else and switch/case
  - loops with for, while, do/while
  - Jump statements return, break, continue

- **But there are important differences! Let's look at the easy stuff first:**

# Class Anatomy

- **The basic block of Java code is a *class***
  - `public` classes can be accessed from code in other packages

```
// This is the file <srcdir>/com/pekia/JellyBean.java

package com.pekia.beans;

public class JellyBean {

    public static final int SUGAR_CALORIES_PER_POUND = 1775;

    private int calories;
    private String flavor;

    public JellyBean(int calories, String flavor) {
      super();
      this.calories = calories;
      this.flavor = flavor;
    }

    public int getCalories() {
      return calories;
    }

    public String getFlavor() {
      return flavor;
    }

    public static void main(String[] args) {
      JellyBean jb = new JellyBean(3750, "TwoPoundPurple");
    }

}
```

- **One `public` class per source file**
  - File name is the same as the class name
  - Package name reflects the directory name of the source and class files

- **C# programmers note**
  - There is no struct keyword; class instances are always reference types.

# Naming conventions

- **These conventions will take a bit of getting used to, but please follow them**

- **package names are lowercase and should start with a reversed domain name identifying the responsible entity**

```
com.apple.quicktime.v2
org.springframework.boot
```

- **Class and Interface names use PascalCasing**

```
BigInteger
String
Object
ProductBeanFactory
```

- **Method, field, parameter and variable names ALL use camelCasing**

```
inputStream
mainWindow
stopDependentThreads()
```

- **constants are UPPERCASE with underscores between words**

```
static final int MAX_ASYNC_QUERIES = 27;
static final double ASPECT_RATIO = 1.75;
```

- **C# programmers note:**
  - Interface names do not have to begin with an uppercase letter I (eye)
  - There are no Property names, because properties are represented in Java by setter and getter methods!
  - You can't use the @ sign prefix to declare an identifier with the same name as a keyword; it is reserved for use with annotations

# Code Formatting Conventions

- **Style is always hotly contested, and you should do what your organization thinks is right:**
  - Tabs vs. spaces for indents
  - Brace alignment and indentation on code blocks
  - Indentation for multi-line statements
  - Whitespace around operators, parentheses, parameters, etc.
  - Comment styles
  - Use of lambdas
  - blah blah blah

- **Best bet - automate this and forget it**

- **In eclipse, set up your preferred formatting style:**
  - Window menu -> Preferences -> Java -> Code Style -> Formatter
  - Use Ctrl-Shift-F in your editor to reformat according to the currently installed style guide

- **Tell git diff to ignore whitespace**
  - Enjoy your life

# Primitive Data Types

- ***Primitive*** **data types are simple boolean, character or numeric values**

- ***All of these are value types***
  - Simple and fast to manipulate
  - Used for most mathematical calculations

| Type | Size (bits) | Value | Range |
|---|---|---|---|
| `boolean` | N/A | logical value | `true` or `false` |
| `char` | 16 | Unicode character | \u0000 to \uFFFF |
| `byte` | 8 | signed integer | -128 to 127 |
| `short` | 16 | signed integer | -32,768 to 32,767 |
| `int` | 32 | signed integer | -2,147,483,648 to 2,147,483,647 |
| `long` | 64 | signed integer | -(2^63) to +(2^63) - 1 |
| `float` | 32 | floating point<br>IEEE 754 Standard | +/- 1.4e-45 to 3.4e38<br>(6-7 significant decimal digits) |
| `double` | 64 | floating point<br>IEEE 754 Standard | +/- 4.9e-324 to 1.8e308<br>(15 significant decimal digits ) |

- **C# programmers note:**
  - All integral types are signed when performing arithmetic operations
  - Size (in bits) is guaranteed the same in all JVMs
  - Primitive type names are all lowercase
  - There is no `sizeof` operator
  - There is no `decimal` type
    - (we use java.math.BigDecimal instead - more later)
  - Not nullable
    - (we use wrapper classes and autoboxing where needed - more later)

# Reference Types

- **All non-primitive types are reference types**
  - All class types

```
Object o = null;
Circle c = new Circle();
Double d = new Double(29.5);
```

  - All arrays

```
int[] lottoNumbers = {1,2,3,4,5,6};

String[] roygbiv = { "Red", "Orange", "Yellow", "Green", "Blue",
                     "Indigo", "Violet" };

double famousNumbers[] = new double[3];
famousNumbers[0] = 3.14;
famousNumbers[1] = 2.73;
famousNumbers[2] = 6.02e23;
```

- **C# programmers note:**
  - Reference types are nullable

# Primitive Wrapper Classes

- **A wrapper class is provided for each of the primitive types**
  - `static` constants define range and special values
  - Methods help to manipulate the type
    - Convert to/from `String` representations
    - Convert between numeric bases (binary, octal, hexadecimal, etc.)
    - Check sign, leading and trailing zeros, shift/reverse bits, etc.

- **`Byte`, `Short`, `Integer`, `Long`**
  - Constants `MIN_VALUE`, `MAX_VALUE`, `SIZE`

- **`Float`, `Double`**
  - Constants `MIN_VALUE`, `MAX_VALUE`, `SIZE`, `MAX_EXPONENT`, `MIN_EXPONENT`, `MIN_NORMAL`, `NEGATIVE_INFINITY`, `POSITIVE_INFINITY`, `NaN`

**Example**

```
int x = Integer.parseInt("23"); // convert String to int

float z = 0.F;
if ( x/z == Float.POSITIVE_INFINITY )
       System.out.println("you divided by zero");
```

- **`Character`**
  - provides useful methods to test the value of a `char`

- **`Boolean`**
  - Constants TRUE, FALSE, NULL

- **C# programmers note:**
  - The wrapper types are reference types, and are all nullable

# Variable declarations

- **All variables require explicit type declarations before Java SE 10**

```
int x;
PrintStream errorStream;
Circle shape;
```

- **Variables may be initialized at the point of declaration**

```
int x = 23;
PrintStream errorStream = System.err;
Circle shape = new Circle();
```

- **As of Java SE 10 you can use implicit typing *for local variables only***
  - The variable must be initialized so the compiler can infer the type

```
var x = 23;
var errorStream = System.err;
var shape = new Circle();
```

  - Though declared with var, every variable has a specific type; if you initialize with a literal value, use the numeric literal suffixes to make sure an integer type is unambiguous

```
var num1 = 12;            // int (the default)
var num2 = 3000123000L;   // long
var flt1 = 98.6;          // double (the default)
var flt2 = 75D;           // double
var flt3 = 2.5F;          // float
```

- **C# programmers note:**
  - Suffix is not case sensitive
  - There is no suffix M (decimal type) or U (unsigned)
  - There is no `dynamic` type

# Constants

- **Variables can be made constant with the `final` keyword**
  - A `final` variable may be assigned to only once
  - Cannot be changed once initialized

**Example**

```
final int MAX_LENGTH = 40;
final double H;
...
H = 6.62517e-27;    // OK to initialize once
MAX_LENGTH = 80;    // Does not compile
```

- **C# programmers note:**
  - the const keyword is reserved but not used in Java.

# Default Initialization

- **`static` member variables (class variables)**
  - Initialized to `0` (zero), `null` or `false`
  - May be initialized dynamically in `static` initialization block

- **non-`static` member variables (instance variables or fields)**
  - Initialized to `0` (zero), `null` or `false`
  - May be initialized dynamically in constructor or non-`static` initialization block

- **Automatic (or "local") variables**
  - *Must* be initialized explicitly before use or won't compile

# Strings

- **A character string is represented by an object of type `java.lang.String`**

```
String s1;     // declare reference variable
```

  - *This is a reference type, but its value is immutable*

- **String literal is a character string in double quotes**

```
s1 = "Hello\tWorld";  // initialize with literal
```

  - Escape sequences can be used in string literals

- **You can create a string object explicitly with the `new` operator**

```
String s2 = new String("World");
```

- **Compare strings with the `equals()` method, NOT with ==**

```
if ( s1.equals(s2) ) {
   System.out.println("s1 and s2 are the same");
}
```

- **Concatenate strings with the "+" operator**

```
String s3 = s1 + " " + s2;
System.out.println( s3 );
```

- **C# Programmers note:**
  - Java does not have interpolated strings, e.g., $"Your name is {name}";
  - Fun fact: the plus sign is the only overloaded operator in Java

# Type Conversion

- **Values may be assigned between variables of *compatible* types**

- **Implicit conversion is allowed if the destination type has a greater range than the source type**
  - Called "widening"

- **Order of widening**

```
byte  ->  short  ->  int  ->  long  ->  float  ->  double
            ^
            |
           char
```

**Example**

```
int i;
float f;
short s = 5;

i = s;  // OK; int is wider than short
f = s;  // OK; float is wider than short
f = i;  // OK; float is (nominally) wider than int
```

- **An `int` *literal* can be assigned to a `byte` or `short` if the value falls within the range of the data type**

# Casting (primitive types)

- **Casting is required to assign a value to a less-precise type**
  - Called "narrowing"

```
s = (short) i;
i = (int) f;
```

  - This may truncate the value if it does not fit!

- **Not allowed between incompatible types**

```
boolean b = true;
i = (int) b;       // does not compile
```

- **`char`, `byte`, and `short` are automatically widened to `int` when used in arithmetic operations**
  - Must cast to assign result back to a type narrower than `int`

```

byte a = 5, b = 6;
a = (byte) (a * b);  // correct
a = a * b;           // does not compile
```

- **Casting a floating-point type to an integer truncates towards zero**

**Example**

```
int i = (int) 4.98;  // i is 4
```

# Autoboxing

- **Sometimes it is useful to treat a primitive value as an object**
  - e.g. when putting it in a `Collection` or an array of `Object`

- **Instances of the wrapper classes can be created with new**
  - A wrapper instance holds an immutable value

**Example**

```
int x = 21;
Integer xWrap = new Integer(x);      // Forever 21
float f = xWrap.floatValue();
String s = xWrap.toString();
x = xWrap.intValue();
```

- **As of Java SE 5, *autoboxing* automatically wraps primitives when the compiler sees that an object is required**

**Example**

```
Integer wrapInt = 8;     // Primitive literals
Double wrapDbl = 4.5;
```

- **Unboxing is also automatic**

**Example**

```
int i = wrapInt + 4; // unboxes before addition
wrapInt++;           // unboxes, increments, reboxes
```

  - The code above does not *change* the value of the `Integer`, but creates a new `Integer` object; the old one may be garbage collected

# Simple Console I/O

- **Use `System.out.println()` to output simple messages**
  - Argument can be a variable, a string in double quotes, or any number of these concatenated with the `"+"` sign.

**Example**

```
double degrees = 75.0;
System.out.println("It is a beautiful, sunny day!");
System.out.println("The temperature is now " + degrees );
```

- **Use `System.out.printf()` to output formatted text**

**Example**

- **Use the `java.util.Scanner` class to read input**

```
import java.util.Scanner;
...
Scanner in = new Scanner(System.in);

String input;

System.out.println("Welcome! What's your name?");
input = in.nextLine();
System.out.printf("hello %s!\n", input);

System.out.println("Tell me, what's your favorite number?\n");
input = in.nextLine();
int x = Integer.parseInt(input);
System.out.printf("I agree; %d is a great number!\n", x);

in.close();
```

- **Java SE 6 provides a `Console` class to facilitate console I/O**
  - Unfortunately, it cannot be used inside of IDEs

# Simple String <-> Number Conversion

- **To convert strings to numbers:**
  - "Wrapper" classes have `static parseXxx()` methods

**Example**

```
int i = Integer.parseInt("1024");
double d = Double.parseDouble("110.50");
```

- **To convert numbers to strings:**
  - `String` class has `static valueOf()` methods

**Example**

```
String s1 = String.valueOf(1024);
String s2 = String.valueOf(100.50);  // yields "100.5"
```

  - An alternate way to convert: concatenate with `""`

**Example**

```
String s3 = "" + 1024;
```

# Section 3-2

# Conditionals & loops

# Control Statements

- **Control statements are similar to those in C#**
  - `if/else`
  - `switch/case`
  - `for` (for counted loops)
  - `while` (pre-test loops)
  - `do/while` (post-test loop)

# `if` Statement

- **The `if` statement allows conditional execution**
  - Statement is executed if the `boolean` test expression evaluates to `true`
  - The optional `else` provides an alternative flow of execution if the test expression is `false`
  - Chained (else if) and nested ifs are fine as long as they are not too complicated for me to read.  P.S. I am not that smart.

**Example**

```
if ( speed < 65.0 ) {
    System.out.println("Speed up!");
} else {
    System.out.println("Slow down!");
}
```

**Example**

```
if (args.length != 2) {
    System.out.println("requires 2 arguments");
    System.exit(-1);
}
```

- **C# programmers note:**
  - As in C#, only a boolean value can be used as a test expression

# Boolean Expressions

- A boolean expression evaluates to `true` or `false`

**Example**

```
if ( yourGuess == pumpkinWeight ) {
  System.out.println("You guessed right!");
}

while ( pageNum < pageMax ) {
   pageNum = printNextPage();
}
```

- **Boolean expressions usually incorporate one or more *relational operators***
  - Relational operators compare values and return a `boolean` value

| | |
|---|---|
| `==` | Equality |
| `!=` | Not equal |
| `<` | Less than |
| `<=` | Less than or equal to |
| `>` | Greater than |
| `>=` | Greater than or equal to |

- **C# Programmers note:**
  - In C# these are called the Equality and Comparison operators
  - *With reference types, the equality operator **always** compares identity; to compare two objects by value, use their equals() method.  This is true for String objects as well.*

# Logical Operators

- **Logical operators are used to combine boolean expressions**

**Example**

```
if ((speed == 0) && (altitude <= groundLevel)){
  System.out.println("We've landed");
}
```

- **Logical operators take `boolean` operands and yield a `boolean` result**

| | |
|---|---|
| `&` | AND – always evaluates both operands |
| `\|` | OR – always evaluates both operands |
| `^` | XOR – always evaluates both operands |
| `&&` | Conditional AND |
| `\|\|` | Conditional OR |
| `!` | Not |

- **Conditional "`&&`" and "`||`" may short-circuit**
  - Expressions are only evaluated until the truth or falsehood of the entire logical expression can be determined

**Example**

```
// If d is zero, second expression is not evaluated
if (d != 0  &&  n / d > 10) { ... }

// if n is 200, count does not get incremented
if (n > 100 || count++ < 3) { ... }
```

# Conditional Operator

- **The *conditional operator* takes three operands (a *ternary* operator)**

```
boolean-expression ? expression1 : expression2
```

  - If `boolean-expression` is `true`, return value of `expression1`, else return value of `expression2`

- **Returns a value that can be used in a surrounding expression**

**Example**

```
int min = (a < b) ? a : b;  // if a < b then a, else b
```

- **Does not evaluate the expression that is not selected**

# Loops

- **`while` executes a statement repeatedly as long as the test expression evaluates to `true`**
  - The test expression is evaluated *before* executing the loop the first time
  - Body of loop is executed *zero* or more times

**Example**

```
int a = 100;
while (a > 0) {
  System.out.println("Counting: " + a);
  a = a - 1;
}
```

- **`do while` evaluates the test expression *after* executing the loop the first time**
  - Body of loop is executed *one* or more times

**Example**

```
char c;
do {
  // Read and process a character
  c = (char) System.in.read();  // read char
  System.out.println(c);        // print char
} while (c != 'X');             // exit loop if 'X'
```

# The for Loop

- **The `for` loop executes a statement repeatedly as long as a test expression evaluates to `true`**

- **The `for` clause holds three expressions, separated by semicolons**
  - *initializer* is executed once, and usually initializes loop variables
  - Body of loop is executed if *continuation_test* is true
  - *modifier* is executed after each iteration, and usually modifies loop variables
  - Any expression in the `for` clause can be omitted
    - If *continuation_test* is omitted, it is always `true` (one way to create an infinite loop)

**Example**

```
for (int i = 0; i < args.length; i++) {
    System.out.println(args[i]);
}

for (;;) {
    System.out.println("work never ends...");
}
```

# Enhanced `for` Loop

- **The enhanced `for` loop, sometimes called the for-each loop, was introduced in Java SE 5**

```
for ( local_variable : array_reference ) {
  // loop statements
}
```

- **It is used to traverse arrays and `Collection` classes without an explicit counter**
  - For each iteration, the next element in order is assigned to *`local_variable`*
  - Statements in loop block are executed with access to the element through *`local_variable`*

- **Loop stops automatically after iterating through all elements**

**Example**

```
package mod05.examples;
public class EchoArguments {
  public static void main(String[] args) {
    for ( String s: args ) System.out.println(s);
  }
}

Output

prompt> java mod05.examples.EchoArguments one two three
one
two
three
```

# Enhanced `for` Loop

- **The enhanced `for` loop (sometimes called the for-each loop) traverses arrays and `Collection` classes without an explicit counter**

- **Loop stops automatically after iterating through all elements**

**Example**

```
String[] roygbiv = { "Red", "Orange", "Yellow", "Green", "Blue",
                     "Indigo", "Violet" };

for (String s: roygbiv) System.out.println(s);
```

- **C# programmers note:**
  - This is the same as the C# foreach statement

# `break` and `continue`

- **`break` exits the containing loop**

**Example**

```
package mod03.examples;

public class BreakExample {
  public static void main(String[] args) {
    // Find the first number evenly divisible by 13
    for (int i = 1; i <= 50; i++) {
      if (i % 13 == 0) {
        System.out.println("i = " + i);
        break;    // Exit the loop completely
      }
    }
    System.out.println("was first divisible by " + 13);
  }
}
```

- **`continue` jumps to the modifier part of a loop**

**Example**

```
package mod03.examples;

public class ContinueExample {
  public static void main(String[] args) {
    // Find all numbers evenly divisible by 13
    for (int i = 1; i <= 50; i++) {
      if (i % 13 != 0) continue;   // Skip rest of loop
      System.out.println("i = " + i);
    }
    System.out.println("were all divisible by " + 13);
  }
}
```

# Labels

- **Code blocks *in loops* can be labeled**
  - Allow programs to `break` or `continue` to the outer portion of a nested loop
  - This *cannot* be used as a general `goto`

- **Label name follows the `break` or `continue` keyword**
  - Without labels, `break` and `continue` work on the immediately containing loop

**Example**

```
outer: for (int i = 0; i < 5; i++) {
         for (int j = 0; j < 5; j++) {
           if (i == j) {
              System.out.print(i + " ");
              continue outer;  // Continue outer loop
           }
         }
       }
```

**Output**

```
0 1 2 3 4
```

- **Makes logic harder to read, so should be used sparingly**

# `switch` Statement

- **Compares value of an expression to a series of constants**
  - *expression* must evaluate to a `byte`, `short`, `int`, `char` or `enum.` Java SE 7 also allows `String` expressions
  - Code after the matching `case` is executed until a `break` is encountered. Multiple cases may execute the same code
  - Cases need not be in numerical order

**Example**

```
for (int a = 0; a < 4; a++) {
    switch (a) {
      case 1:
      case 2:
        System.out.println("a is one or two.");
        break;
      case 0:
        System.out.println("a is zero.");
        break;
      default:
        System.out.println("a is greater than 2.");
    }
}
```