
Getting Started with Java



This chapter introduces the Java platform. We describe what "Java" actually means, and show how to get started writing simple Java applications.

We'll start off by using the Java JDK (Java Development Kit) tools manually from the command line. For example, we'll show how to use `javac` to compile a Java application, and how to use `java` to run a Java application. Then we'll show how to use Eclipse, a popular and free Java Integrated Development Environments (IDEs)

Contents

1. Setting the scene
2. Installing and using Java SE
3. JAR files
4. Using an IDE



Demo folder: `DemoGettingStarted`

2

Each chapter in this course is divided into several sections. There are four sections in this chapter...

Section 1 sets the scene. We introduce the Java platform, and describe the differences between Java Standard Edition (Java SE) and Java Enterprise Edition (Java EE).

Section 2 shows how to install the Java SE. It's simple and free, and we've actually already set everything up on your computers already. Nonetheless it's worthwhile knowing how to get Java SE for yourself, because you'll doubtless have to do this at some stage in your Java programming career.

Section 3 discusses JAR files. A JAR file is a zip file that contains compiled Java code. You can use JAR files as the standard way to distribute your Java applications to other developers and users.

Section 4 describes the benefits of using an Integrated Development Environment (IDE) rather than using the Java JDK from the command line. We'll show how to download and use Eclipse for Java SE.

Note: Each chapter in this course has a dedicated set of demo code and lab code. The demos for this particular chapter are located in the `DemoGettingStarted` sub-folder. So, if the course code is installed in the standard location, you'll find the demos for this chapter here:

- `C:\JavaDev\Demos\DemoGettingStarted`

1. Setting the Scene

- Hello Java
- Java timeline
- Java vs. other languages
- What can you do with Java SE?
- What can you do with Java EE?



It's time to get started. This section outlines the principles of Java, so you have a mental context for the details that follow.

Hello Java

- Java is one of the most widely-used object-oriented languages in contemporary software development
 - Other popular OO languages include C#, C++, etc.
- OO principles:
 - Abstraction
 - Encapsulation
 - Inheritance
 - Polymorphism

4

Java emerged in the mid 1990's and has grown in scope, popularity, and complexity over the last two decades. Java is fully object oriented:

- You define classes to represent abstractions of the real world. For example, you can define classes such as `Person`, `BankAccount`, `Transaction`, `Payment`, and so on. A class specifies data plus a related set of operations. For example, a `BankAccount` class might define data members such as `balance` and `accountHolder`, and operations such as `deposit()` and `withdraw()`.
- You use encapsulation to keep details as private as possible within your classes. For example, you define the data values private inside a class, so that other parts of the application can't access the data directly. This helps you create more robust code, which you can modify more safely without worrying about breaking other parts of the application.
- You can use inheritance to define a new class based on an existing class. This enables you to reuse existing classes, so that you can define a new class more quickly in terms of how it differs from an existing class. We call the existing class the "superclass", and the new (variant) class the "subclass".
- Polymorphism is related to inheritance. It's a mechanism that allows you to create objects for different subclasses (e.g. different kinds of bank accounts), where each subclass implements certain operations differently (e.g. different rules for withdrawing money). Via polymorphism, you can use all these objects in a consistent manner, without knowing what type of object it is.

Java Timeline

- Java first appeared in 1996
 - Java Development Kit (JDK) 1.0
 - Contained 6 packages
 - Original goal was to create platform-agnostic apps for mobile devices
 - In practice, many organizations used Java to create applets
- Sun reorganized their Java products in 1999
 - Java 2 Standard Edition (J2SE)
 - Java 2 Enterprise Edition (J2EE)
 - Java 2 Micro Edition (J2ME)
- There have been many subsequent upgrades to the language and libraries!

5

When Java first emerged, it had a very small set of standard classes. There were just 6 packages (a package is a group of related classes, e.g. the `java.io` package contains classes that perform file I/O).

Since its first appearance, Java has grown into a massively popular language for creating all kinds of application. For example:

- Android applications for mobile devices are written in Java.
- Server-side applications for generating web pages and for implementing web services can be written in Java (or C#, or PHP, etc.)
- Utility applications for file handling, command-line tools etc. can be implemented in Java (or Perl, or PowerShell, etc.)

Oracle are the current owners of the Java specifications. This course uses Java Standard Edition (SE) version 7.

Java vs. Other Languages

■ Java vs. C++

- Java syntax is very similar to C++ (not coincidental)
- Java compiles to platform-neutral byte codes that run on a Java Virtual Machine (JVM), whereas C++ compiles to platform-specific machine instructions
- Java handles memory management via garbage collection, whereas C++ makes you manage memory yourself
- Java has an extensive and rich class library, whereas the C++ standard class library is much more limited (until C++11 recently)
- Java doesn't run as quickly (or as leanly) as C++

■ Java vs. C#

- Java syntax is very similar to C# (again, this is no coincidence!)
- In fact, Java and C# are quite comparable in many aspects...

6

When Java first came out, the dominant OO language at that time was C++. So, not unsurprisingly, the Java syntax is strongly reminiscent of C++ syntax.

However, there are some significant differences between Java and C++, as noted on the slide above. The key difference is that Java code compiles to intermediate byte codes, which are Just-In-Time (JIT) translated into machine code instructions by a virtual runtime engine called the Java Virtual Machine (JVM). The JVM is a standard part of the Java JDK.

At the turn of the millennium, Microsoft launched the .NET platform and with it came the C# programming language (plus VB.NET). C# is also rather similar to Java, and generally programmers find it quite an easy transition moving from one language to another. Having said that, the Java SE class library is completely different to the .NET Framework class library, so that's one area you'd need to concentrate on if you ever do find yourself moving from Java to C# (or vice versa).

What can you do with Java SE?

- You can use Java SE to:
 - Create standalone Java applications
 - Create Windows-based applications
 - Manipulate text files, binary files, and XML data
 - Access databases, via Java Database Connectivity (JDBC)
 - Create multithreaded applications
 - Create client-server applications, using Remote Method Invocation (RMI)
 - Etc...



This slide summarizes the main capabilities of the Java SE platform. The Java SE class library contains a large number of classes that allow you to perform these tasks, plus much more beside.

The library is large and sometimes rather complex. As your Java programming career develops, you'll find yourself gaining more experience in the relevant nooks and crannies that are relevant to your needs. Don't feel obliged to sit down with a large API book and try to learn the libraries by rote!

What can you do with Java EE?

- Going further, you can use Java EE to create:
 - Web applications, using servlets, JavaServer Pages (JSPs), and Java Server Faces (JSF)
 - Web services
 - Middle-tier session beans, using Enterprise JavaBeans (EJBs)
 - Data-access entity beans, using the Java Persistence API (JPA)
 - Message-queueing applications, using the Java Message Service (JMS)
 - Etc...

8

Java EE is a layer on top of Java SE. Java EE provides a large number of additional classes (plus several additional command-line tools) that enable you to create enterprise applications, including:

- Web applications, using a variety of APIs.
- Web services, both SOAP-based and REST-based.
- EJB components, which benefit from pooling, declarative security, transaction management, etc.
- Object persistence, via the JPA API.
- Message-based systems, by sending/receiving messages to a queue or by publishing/subscribing to messages in topics.

Note: Java EE is outside the scope of this course. We cover Java EE in detail in our *Practical Java EE* course.

2. Installing and Using Java SE

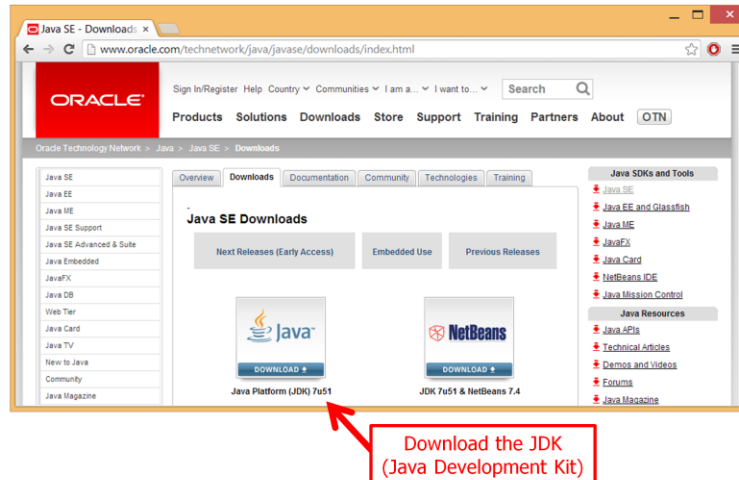
- Downloading Java SE
- Installing Java SE
- Java SE folder structure
- Minimalistic development

9

This section describes how to download and install the Java SE. We also show how to write a simple "Hello World" Java application, compile it, and then run it from the command line.

Downloading Java SE

- Download Java SE (details vary depending on version) :
 - <http://www.oracle.com/technetwork/java/javase/downloads/index.html>



Oracle are the current owners and custodians of Java standards and products. Almost everything Java-related is free.

The computers on this course already have Java SE and Eclipse installed, but it's worthwhile describing how to set everything up from scratch anyway, because you'll have to do it yourself at some point.

You can download Java SE from the following web site:

- <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

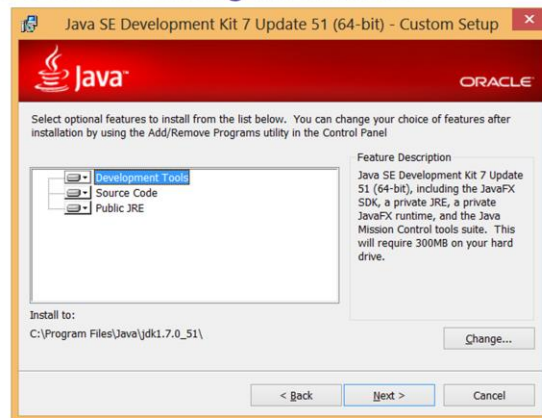
The screenshot in the slide above shows this web site (your screenshot might differ, depending on version numbers etc.). Click the link to download the Java Platform (JDK). We're using the JDK version 7 on this course. There is also a JDK version 8 scheduled for release in Spring 2014, which includes various language extensions.

Note: The download page differentiates between the JDK and JRE:

- The JDK is the Java Development Kit, and includes command-line tools such as the Java compiler, Java debugger, JAR tool, etc. If you want to develop Java applications (as well as run them, obviously), this is what you need.
- The JRE is the Java Runtime Environment, and includes the minimal set of tools you need to run Java applications. It's a subset of the JDK. If you just want to run Java applications (i.e. not develop them), this is all you need.

Installing Java SE

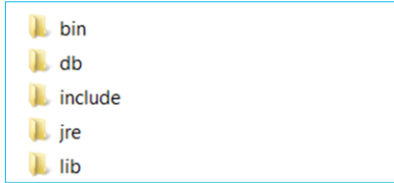
- Select and run the appropriate JDK installation
 - E.g. `jdk-7u51-windows-x64.exe`
- Accept all defaults during installation



Make sure you select the correct JDK installation for your target computer, e.g. Linux, Solaris, Windows 32-bit or 64-bit.

When the JDK download is complete, just run it and accept all the default options during installation.

Java SE Folder Structure

- Default installation folder for Java SE (v1.7, update xx):
 - C:\Program Files\Java\jdk1.7.0_xx
- Folder structure:

The screenshot shows a list of five sub-folders within the Java SE installation directory: bin, db, include, jre, and lib. Each folder is represented by a yellow folder icon and its name is listed to the right.

12

The screenshot in the slide above shows the installation folder for the JDK. Here's a brief description of each sub-folder:

- `bin`
Contains the JDK tools (compiler, etc.)
- `db`
Contains a Java-based database engine called Derby. We'll be using Derby in the database chapter later in the course.
- `include`
Contains C/C++ header files if you want to integrate your Java code with native C/C++ code via JNI (Java Native Interface). JNI is out of the scope of this course. We cover it in our *Advanced Java Development* course.
- `jre`
Contains JRE tools (JVM, etc.)
- `lib`
Contains various library used by the JDK tools.

Minimalistic Development (1 of 2)

- Define a public Java class
 - Filename must be `classname.java`

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

MyFirstApp.java

- Compile the class from the command line
 - Using the Java compiler, `javac.exe`
 - Note, `javac.exe` must be on the path

```
javac MyFirstApp.java
```

→ Creates MyFirstApp.class

13

This slide shows a traditional "Hello World" application, as implemented in Java. The first code box shows `MyFirstApp.java` – you can find this file on your computer, in the demos folder for this chapter.

We won't worry too much about the syntax yet – it should look reasonably intuitive. We'll just point out three main points at this stage:

- `MyFirstApp.java` defines a public class named `MyFirstApp`. Everything in Java must reside inside a class, because Java is completely OO. You can't define anything outside of a class (unlike in C or C++, where you can define global variables and functions).
- A Java file can only contain a single public Java class, and the class name must correspond to the file name.
- The class has a `main()` method, which is the entry point for this application. The `main()` function simply displays a message on the console, via the `System.out.println()` function.

Before you can run this class, you must first compile it via `javac` (i.e. the Java compiler). You can run this from the command line – but be sure `javac` is on the system path. The Java compiler compiles `MyFirstApp.java` into a binary file named `MyFirstApp.class`. The binary file contains Java byte codes, i.e. compiled Java instructions.

You cannot run `MyFirstApp.class` directly on your operating system – you must run it through the JVM instead, as described on the next slide.

Minimalistic Development (2 of 2)

- Run the class from the command line

- Via the JVM, `java.exe`
- Note, `java.exe` must be on the path

```
java MyFirstApp  
Hello world
```

Note, you specify the classname
without the `.class` file extension

14

To run a Java application, you must always run it through the JVM. To do this, run the `java` application and specify the name of your class file (without the `.class` file extension).

The `java` application is the JVM. The JVM loads your class definition into memory, and JIT-compiles the Java byte codes into native machine instructions for your particular target computer.

There are ports of the JVM to a wide range of operating systems and platforms. This means you can run the same Java code on many platforms without recompilation – all you need is an appropriate JVM for your particular platform.

3. JAR Files

- Overview of JAR files
- How to create a JAR file
- Example
- How to run a JAR'd application

15

This section introduces you to the concept of JAR files. We describe what a JAR file is, and then show how to create and use a JAR file.

JAR files are the standard way to bundle a group of related Java classes into a single file, to make it easier to deploy and reuse the classes elsewhere in your system.

Overview of JAR Files

- In a real Java application, you'll have lots of Java classes
 - Each class is compiled to a separate `.class` file
- It's common to bundle up all the classes etc. into a JAR file
 - Typically use the `.jar` file extension
 - The Java Archive (JAR) file format is a zip-file really
- You can manage JAR files by using the `jar.exe` tool from the command line
 - Part of the JDK 😊

16

JAR files are a very important part of Java development. A JAR file is a zip file that contains compiled Java classes and other related files, e.g. images, XML documents, etc.

JAR files are the standard mechanism for creating and sharing libraries in Java. Whenever you use a third-party library, e.g. to access a vendor-specific database or to use a graphics library, the library is published as a JAR file.

JAR generally files have a `.jar` file extension. There are also some special-purpose JAR files that have different file extensions, for example you can create `.war` files to hold web applications, and you can create `.ear` files to hold Java EE applications that comprise web components, EJB components, etc.

How to Create a JAR File

- To create a JAR file:

```
jar cfm jar-file-name manifest-file-name input-file(s)
```

- Here's a description of the options:

- c means you want to create a JAR file
- f means you want the output to go to a file (rather than STDOUT)
- m means you want to include a manifest file (see later)

- And here's a description of the file names:

- jar-file-name is the name of the JAR file you want to create
- manifest-file-name is the name of the manifest file (see later)
- input-file(s) is a list of files to include in the JAR file

17

To create a JAR file, you can use the `jar` utility in the JDK. The `jar` utility takes a range of command-line parameters to specify exactly what you want to do. Consider the following example:

```
jar cfm MyLibrary.jar manifest.txt *.class
```

This will create a JAR file named `MyLibrary.jar`. The JAR file will contain a manifest file (discussed later) named `manifest.txt`, and will contain all the `.class` files in the current folder.

Example: Java Files

- Let's see how to create a JAR file containing 2 Java classes
 - See the DemoGettingStarted folder
- Here's the "main" Java class for the application:

```
public class MyProg {  
    public static void main(String[] args) {  
        System.out.println("Hello from my program.");  
        MyHelper.displayDateTime();  
        System.out.println("Goodbye from my program.");  
    }  
}
```

MyProg.java

- And here's a "helper" Java class for the application:

```
public class MyHelper {  
    public static void displayDateTime() {  
        System.out.println("It is now " + new java.util.Date());  
    }  
}
```

MyHelper.java

18

The examples in the slide will help you understand how to defined and use a JAR file. You can find these source files in the demos folder for this chapter.

The upper code box shows the MyProg class, which defines the main entry point for the Java application. The MyProg class makes use of the MyHelper class, which is defined in the lower code box.

Example: Specifying the Entry Point

- If you have an application bundled in a JAR file, you need some way to indicate which class file is the entry point
 - You provide this information in a "manifest file"
 - Set the `Main-Class` property to the name of the main class

```
Main-Class: MyProg
```

```
Manifest.txt
```

- Notes:
 - You must specify the fully-qualified name of the class (including its package name - see later for more info about packages)
 - The manifest file must end in a new line or carriage return

19

A JAR file can contain a special file called a manifest file. The manifest file can be named anything you like (e.g. `manifest.txt` in the slide above).

The purpose of the manifest file is to contain additional metadata about the contents of the JAR file – for example, which of the (potentially many) classes in the JAR file contains the `main()` method. The slide shows how to achieve this.

Example: Creating the JAR File

- First, compile the Java classes:

```
javac MyProg.java MyHelper.java
```

- Then create a JAR file containing the .class files and the manifest file

```
jar cfm MyFirstJarFile.jar manifest.txt MyProg.class MyHelper.class
```

20

To create a JAR file, follow these steps:

- Compile all the .java files into .class files. The first box in the slide shows how to do this.
- Create a JAR file that contains the .class files, plus the manifest file. The second box in the slide shows how to do this.

How to Run a JAR'd Application

- To run a JAR'd application:
 - Run `java.exe` to launch the Java Virtual Machine
 - Use the `-jar` option to indicate you want to run a JAR file
 - The JVM consults the manifest file to determine the main class

```
java -jar MyFirstJarFile.jar
Hello from my program.
It is now Mon Feb 17 10:54:07 2014
Goodbye from my program.
```

21

After you've created the JAR file, you can run the application that is contained inside the JAR file. To do this, run `java` (i.e. the JVM) and use the `-jar` command-line option to indicate which JAR file you want to run. The JVM will locate the manifest file inside the JAR file to identify the main class, and will then run the main class. Thereafter, everything unfolds as normal.

4. Using an IDE

- Popular Java IDEs
- Downloading and installing Eclipse
- Starting Eclipse
- Loading existing projects
- Creating a Java project
- Adding a class
- Implementing the class
- Running the class

22

Now that you know how to run the basic JDK tools from the command line, we're going to shift gear and show how to use an IDE.

In practice, all Java developers use an IDE (rather than a text editor and the command line). Using an IDE offers the following benefits:

- Improves productivity
- Simplifies compile/run cycle
- Offers IntelliSense
- Provides code refactoring capabilities
- Simplifies library management

Popular Java IDEs

- Popular Java IDEs:
 - Eclipse (we'll be using this)
 - Oracle JDeveloper
 - IBM WebSphere Application Developer
 - NetBeans
 - Etc...

23

We're going to use Eclipse for Java SE on this course. It's free and extremely extensible via plug-ins.

Other IDEs are available. The slide lists some popular choices. Not all these IDEs are free, although this won't be an issue for you if your company has purchased a site license for the product.

Downloading and Installing Eclipse

- In a browser window:
 - Browse to <http://www.eclipse.org/downloads>
 - Click the Eclipse IDE for Java Developers product
- In the next Web page:
 - Click the appropriate download link for your platform, e.g. Windows 64-bit
 - The download is a simple zip file
- Unzip the Eclipse zip file
 - To the C:\ destination folder
 - Creates a folder named C:\eclipse

24

Note: This slide describes how to download and install Eclipse. This is for background information only – we've already installed Eclipse on your computer for this course.

Eclipse offers several different IDEs, aimed at different programming languages and development targets.

We'll be using Eclipse IDE for Java Developers, which basically means it targets Java SE. Note that there's also an Eclipse for Java EE, which includes additional wizards and libraries for building enterprise applications.

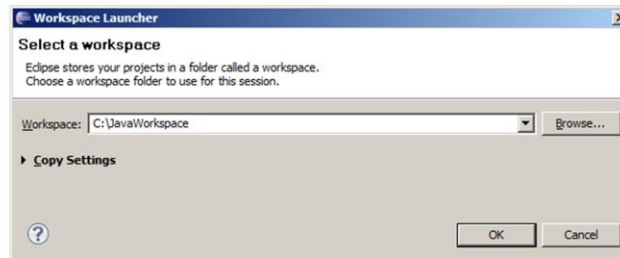
Every so often, Eclipse publish a new version of the IDE. These versions are named Galileo, Helios, Indigo, Juno, Kepler, etc. The next version will start with an L, then an M, and so on. Generally, you'll want to use the latest version available.

Make sure you download the appropriate file for your platform, e.g. Linux, Windows 32-bit, or Windows 64-bit. Then choose a mirror site to download from. The download is a simple zip file, which you can put anywhere on your local computer.

When you unzip the file, it will create a folder named `eclipse`, which contains an executable file named `eclipse.exe`. This is the Eclipse IDE.

Starting Eclipse

- Start Eclipse
 - i.e. run `c:\eclipse\eclipse.exe`
- Eclipse prompts you for a workspace
 - The workspace is a folder that will contain all your Java projects
 - Specify `C:\JavaWorkspace`

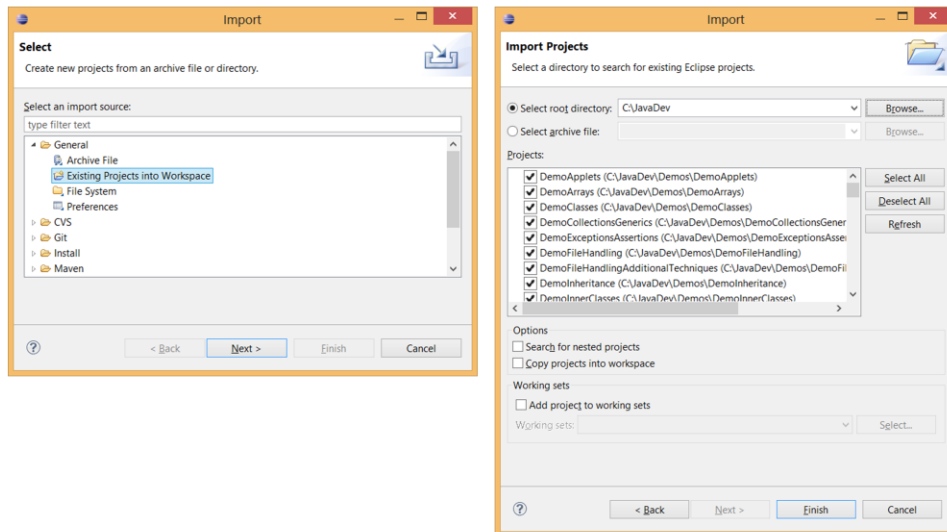


25

When you run the Eclipse IDE, you will be asked to select a workspace. A workspace is a folder where you will put your Eclipse Java projects. You should enter a workspace name such as `C:\JavaWorkspace`.

Loading Existing Projects

- You can load existing projects into the workspace



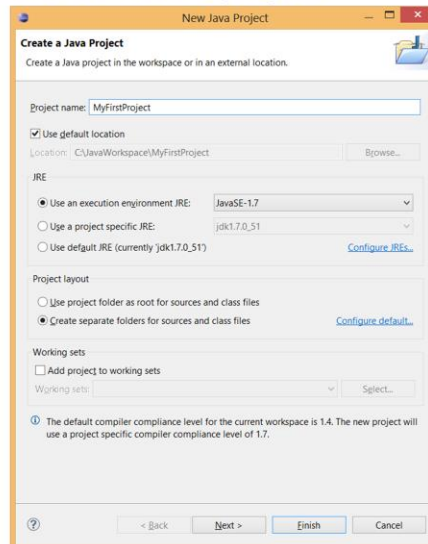
You can load existing projects into an Eclipse workspace as follows:

- On the File menu, click Import.
- In the Import dialog box, expand the General node and select the Existing Projects into Workspace item. Then click Next.
- In the Import Projects dialog box, click Browse and browse to the folder that contains your Java projects (e.g. C:\JavaDev). The dialog box should show all the demo and lab projects for the course.
- Also notice there's a Copy Projects into Workspace checkbox. We recommend you leave this unchecked, so that the workspace *points* to the original project folders (rather than *copying* the projects into the local workspace folder, which can lead to confusion because you'll have two copies of your code).

After you close the Import Projects dialog box, Eclipse should show all the projects in the Package Explorer pane on the left-hand-side of the IDE.

Creating a Java Project

- You can create a new Java project



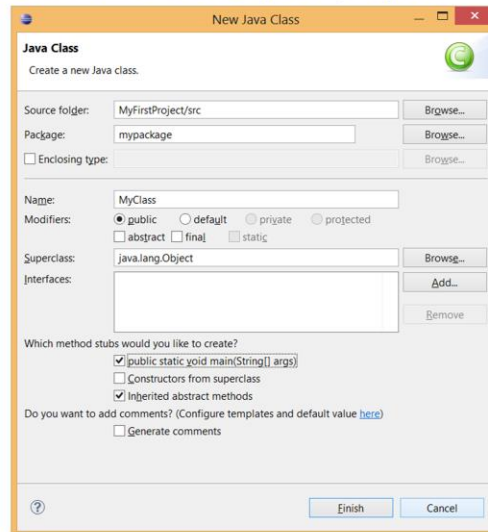
You can create a new Java project in the workspace as follows:

- On the File menu, click New, then click Java Project.
- In the New Java Project dialog box, enter a meaningful project name such as `MyFirstProject`. You can accept all the other default options, e.g. the JRE execution environment should point to your Java 7 JRE. Click Finish to close the dialog box.

After you close the New Java Project dialog box, your project should now appear in the Package Explorer pane.

Adding a Class

- You can add a new class to your project



You can create a new Java class in your project as follows:

- Right-click the src folder. In the popup menu, click New, and then click Class.
- In the New Java Class dialog box, enter a meaningful name for the class (e.g. MyClass). Class names always start with an uppercase letter in Java.
- Enter a meaningful name for the package (e.g. mypackage). You should always put your Java classes into a package. Package names are always lowercase in Java.
- Notice there's a checkbox that enables you to auto-generate a main method for your class. Select this option, to save yourself some typing!

After you close the New Java Class dialog box, the class should appear in the code editor pane in the IDE.

Implementing the Class

- Implement the class, e.g:

```
package mypackage;  
  
public class MyClass {  
  
    public static void main(String[] args) {  
        System.out.println("Eclipse says Hello!");  
    }  
}
```

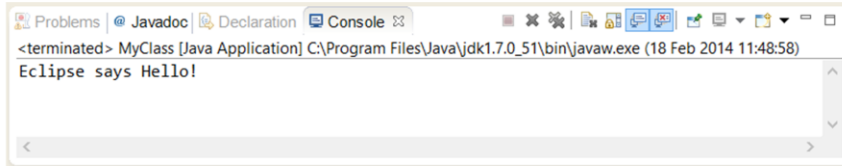
MyClass.java

29

This slide shows a simple implementation for MyClass. The main point at this stage is just to get something working!

Running the Class

- Run the class and see the output in the console window



To run the class using Eclipse:

- Right-click in code editor pane, click Run As, and then click Application. Note that Eclipse has access to the JDK and classpath, so it's quite capable of compiling and running Java applications.
- The application runs and displays its output in console window.

Eclipse also lets you debug a class. Set breakpoints anywhere in the code by double-clicking in the left-hand-side margin next to the code, and then run the class in debug mode via the Debug As | Application menu option.

Any Questions?



31