

# Chapter 2

## Compiling and Running Java Programs

Foundational Java

Key Elements and Practical Programming

# Objectives

- Gain an understanding of the Java compiler and runtime environment
- Learn how to use packages and the classpath
- Learn to use Java from the command line
- Gain some experience with the Eclipse Java Integrated Development environment (IDE)

# Java From the Command Line

- For our first couple of examples we will be compiling and running our Java code from the command line
- Useful to at least get some idea of what is happening in the background when you use a Java IDE
- Understand what happens when
  - code is compiled
  - how it is run
  - how packages and the Java classpath relate to the file system
- Can help to solve problems in development and deployment

# Java Environment

- Check that the **path** environment variable includes the JDK's bin folder (not the JRE's bin folder!)
- On Windows systems, the default location is usually something like:

```
C:\Program Files\Java\jdk1.7.0\bin
```

- We have to be aware of the classpath variable too
  - Can be set as an environment variable and/or in a command window

# Command Window

- If you don't have Windows 7, you can add the Command Window to the context menu in File Explorer (Windows XP)
  - In File Explorer, open **Tools -> Folder Options**.
  - Select the **File Types** tab
  - Select the entry labelled **[NONE] Folder**
  - Press the **Advanced** button and select **New**
  - In the **action** block, type `Command Prompt`
  - In the **application used to perform action** block type `cmd.exe`.
  - Save and exit Folder Options.
  - Now in Windows Explorer you can right click on any folder, select **Command Prompt** and a command window opens in that folder.

# A First Java Program

- Here is a simple Java program

```
public class MyJavaProgram
{
    public static void main(String[] args)
    {
        System.out.println("My Java Program Running!");
    }
}
```

Let's dissect it...

# The Class Declaration

- The first line of code (after the comment header) declares the class

```
public class MyJavaProgram
```

- No Java code can be written that does not belong to one class or another.
- The naming conversion for class names is *Pascal Case*
- The 'public' prefix means that the class can be visible to all other classes, even those that are not in the same *package*.
- We are not specifying which package this class is in, so it will be put into the unnamed *default package*

# The Java File

- The MyJavaProgram class must be saved in a file called 'MyJavaProgram.java'
  - Same mix of upper and lower case letters.
- Create your Java source code
  - Can use any text editor to write your java code
  - Save the file using the .java suffix



# Compiling Java

- Check that javac is available on the system path
- From a command window, compile the source code into byte code, using javac
  - i.e.

```
javac MyJavaProgram.java
```
- You must use the full filename, including the '.java' extension.
- If successful (i.e. no error messages) this will produce the file MyJavaProgram.class in the same folder

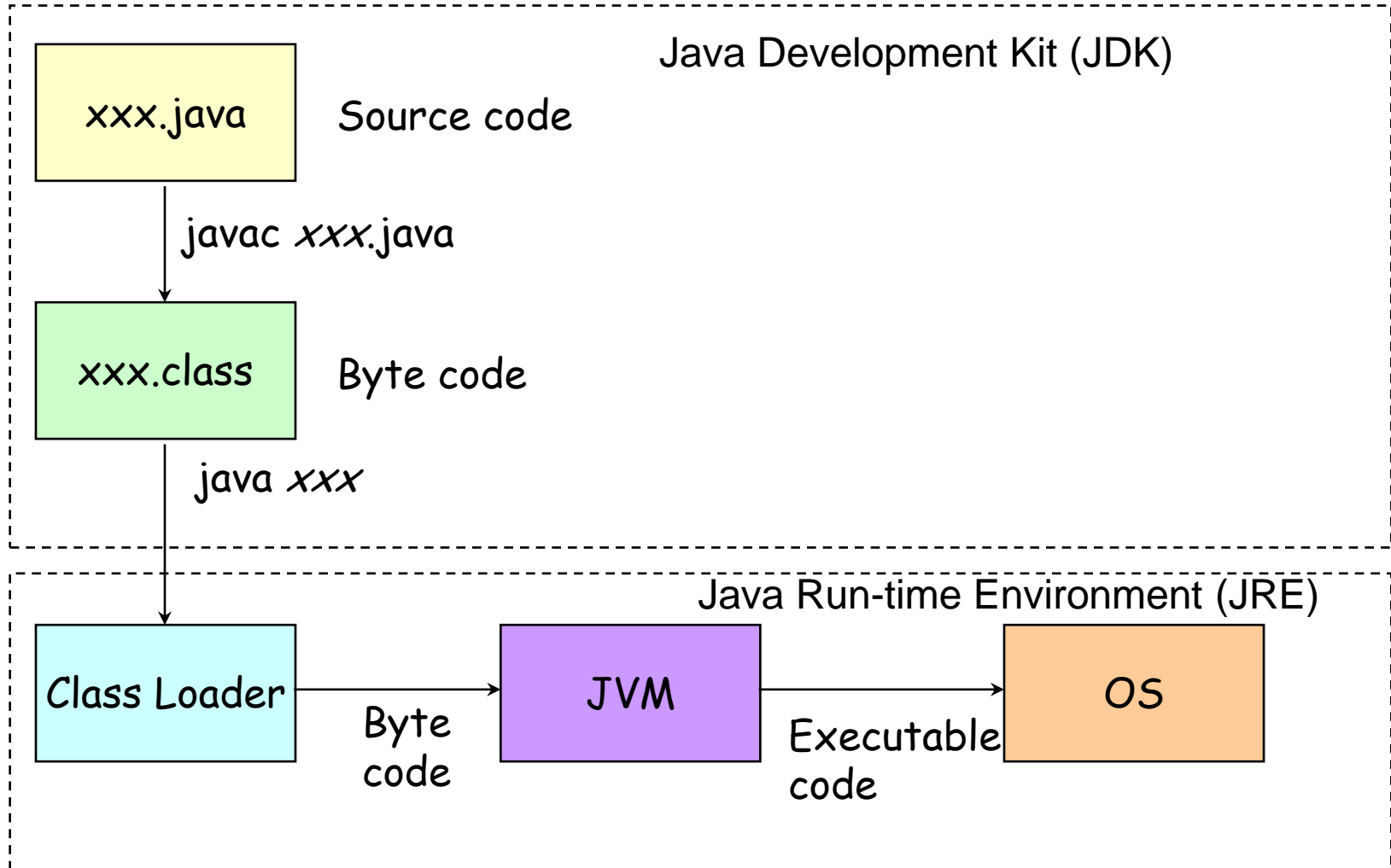
# Running Java

- Run the byte code on the JVM (the virtual machine is case sensitive)

```
java MyJavaProgram
```

- The Java Virtual Machine (JVM) runs the class file `MyJavaProgram.class`
  - But don't use this extension with `java.exe`
- It must be able to find this file on the classpath
- By default, the classpath is the current directory

# Java Build and Run Process



# MyJavaProgram Exercise

- Enter the MyJavaProgram class source code into a text editor
- Save it in a file called `MyJavaProgram.java`
- From the source code folder in the command window, type

```
javac MyJavaProgram.java
```

- This will compile your code
- Fix any errors, and try again
- Run the program by typing

```
java MyJavaProgram
```

# The Classpath

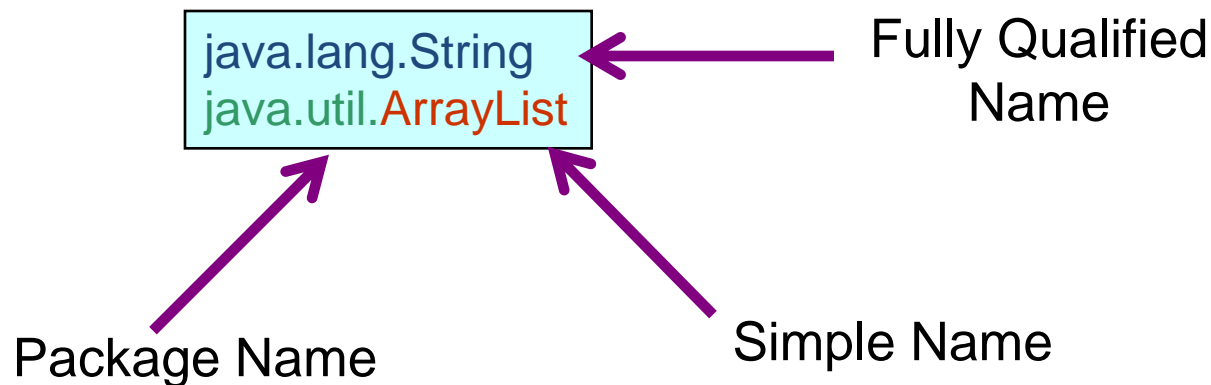
- The Java classpath is what the Java compiler and virtual machine use to find compiled .class files (byte code)
- By default, the classpath is the current directory
  - This is why our example program ran without us being aware of the classpath
  - The JVM found the class file in the current directory
  - However, we can't always work with everything on one directory, so we need to understand how to use packages and the classpath

# Packages

- All Java classes are placed in a package, even if it is the default (unnamed) package
- A package is a bundle of classes
  - The classes in a package are typically related by function
- A package is a name space
  - Essential for distinguishing different classes with the same name
- Each class must have a unique name in the package
- Packages are named hierarchically
  - `java.lang`
  - `java.util`
  - `com.introjava.examples`

# Naming Rules

- Fully qualified class names have two components
  - Package Name
    - all lower case
  - Simple Name
    - Pascal case



# java.lang

- Java platform packages begin with ‘java’ or ‘javax’ (Java Extension)
- Core Java types (like Object and String) are found in the java.lang package
  - This package name does not have to be explicitly used

optional

```

java.lang.Object myObject;
Object anObject;
String aString;
    
```



# Storing Packages in Directories

- Classes belonging to the same package are stored in the same directory
  - The directory is named after the package
- Periods in the package name are replaced by directory separators
- e.g. a class in `com\introjava\examples` would be in the `com.introjava.examples` package
- The Java source can be anywhere
  - It's the compiled class files that must be in the folder that matches their package

# Specifying the Package of a Class

- To specify the package that a class belongs to, use the “package” keyword followed by the name of the package
  - Place this information on the first line of the file
  - Ends with a semicolon
- A class can only belong to one package
  - If no package is specified, the “default” package is used (the current folder)

```
package com.introjava.examples;  
public class MyJavaProgram  
{...
```

# Compiling into a package

- By default, the Java compiler puts class files in the same folder as the Java source
  - This is only OK if the Java source is in a folder that matches its package name
- Otherwise, use the `-d` option on the Java compiler

e.g. current folder

```
javac -d . MyJavaProgram.java
```

- This builds the package folder structure from the given directory

# Classpath and Directories

- The classpath must start from the root of the package structure
  - This may not be the root of the drive
- For example, if the package is `com.softed.examples`, in this folder structure:
  - `C:\javacode\com\introjava\examples`
- Then the classpath must start in the `javacode` folder:

```
SET CLASSPATH=C:\javacode
```

# Running the Class

- The full name of a Java class is its package name followed by the class name, e.g.
  - `com.introjava.examples.MyJavaProgram`
- When you run Java programs you must use the full name of a class with a main method, e.g.

```
java com.introjava.examples.MyJavaProgram
```

# Comments

- C style:
 

```
/*
 * Anything between the
 * slash-star and star-slash
 * is a comment
 */
```
- C++ style
 

```
//single line comments
```
- Javadoc
  - Use as class headers and method descriptions
 

```
/**
 * Anything between the
 * slash-star-star and star-slash
 * is a documentation comment
 */
```

# Exercise

- Make a copy of your MyJavaProgram class
- Rename both the class and the file 'MySecondProgram'
- Add an appropriate 'package' statement to the top of the source file
  - Use the reverse URL convention
- Add comments
- Change the message in System.out.println so you can be certain which class you are running
- Compile the class with the -d option to create the package folder structure
- Set the classpath on the command line (or as a system variable)
- Run the program (remember you need the fully qualified class name)

# Using Eclipse

- Now we will rewrite the previous example using tools within the Eclipse IDE
  - Create a Java project
  - Add a new package
  - Add the class to the package with a 'main' method
  - Add the required code to the main method
  - Save the file (automatically compiles)
  - Run the class