Getting Started

Introduction to Java

Alan Hohn
Alan.M.Hohn@lmco.com

25 June 2013

Contents

Course Purpose

- This course is an introduction to Java
- Assumes programming experience, but little or no Java background
- Tries to prepare software engineers for real-world Java
  - Large-scale applications
  - Distributed communications
  - Commonly-used Java build tools
- Originally delivered to help a team of about 30 experienced software engineers who were new to Java and had a large-scale application to build

Course Contents

- Getting started writing Java programs (today)
- Java programming language basics (4 sessions)
- Packaging Java programs (1 session)
- Core library features (6 sessions)
- Java user interfaces (2 sessions)

Java Portability

- A key motivation for Java is portability (ability to run on different platforms without recompiling)
- This is accomplished by compiling for a "virtual machine" with its own instruction set
  - Known, appropriately, as the Java Virtual Machine (JVM)
  - JVM instructions are called "bytecode"
  - Looks like assembly / machine language, but with added features like virtual function calls
- The JVM provides a way to run bytecode on a specific operating system / machine instruction set
- Generally, the same bytecode can be run unmodified on any JVM

Interpreted vs. Compiled

- Java is not really an interpreted language
  - Java source code must be compiled to bytecode before it can be run
  - Bytecode is sometimes translated on-the-fly to machine instructions
  - Most JVMs provide a Just-In-Time compiler so "hotspots" are compiled down to machine instructions for speed

Java vs. JVM

- Java is a programming language
- The JVM is a general-purpose environment for running bytecode
- In theory, any language could be compiled to bytecode
- There are many scripting languages that can be run on the JVM
  - Groovy, Ruby (via JRuby), Scala, etc.
  - http://en.wikipedia.org/wiki/List_of_JVM_languages
- It's important to understand the distinction between source code (e.g. Java) compiled to bytecode and a scripting language with a runtime interpreter in the JVM

JRE vs. JDK

- JVMs typically come in two different packages
- The Java Runtime Environment (JRE) provides just the JVM plus some core libraries
- The Java Development Kit (JDK) provides a JRE plus the Java compiler (javac), documentation tools, and various other useful things
- Integrated Development Environments (IDEs) such as Eclipse often also include a built-in Java compiler, debugger, or other tools

Hello World in Java

```java
// Java source code is organized into packages
package org.anvard.introtojava;

/* Java is an object oriented programming language.
 * All source code must be organized into "classes"
 * (discussed next time) */
public class HelloWorld {

  /* Note the C-like syntax. Like C, the first code
   * run in a Java program is in a function called main */
  public static void main(String[] args) {
    /* The dot notation is used to find classes
     * inside packages, and items inside classes.
     * This means "find the method called println in the
     * object called 'out' in the class called System. */
    System.out.println("Hello world");
  }
}
```

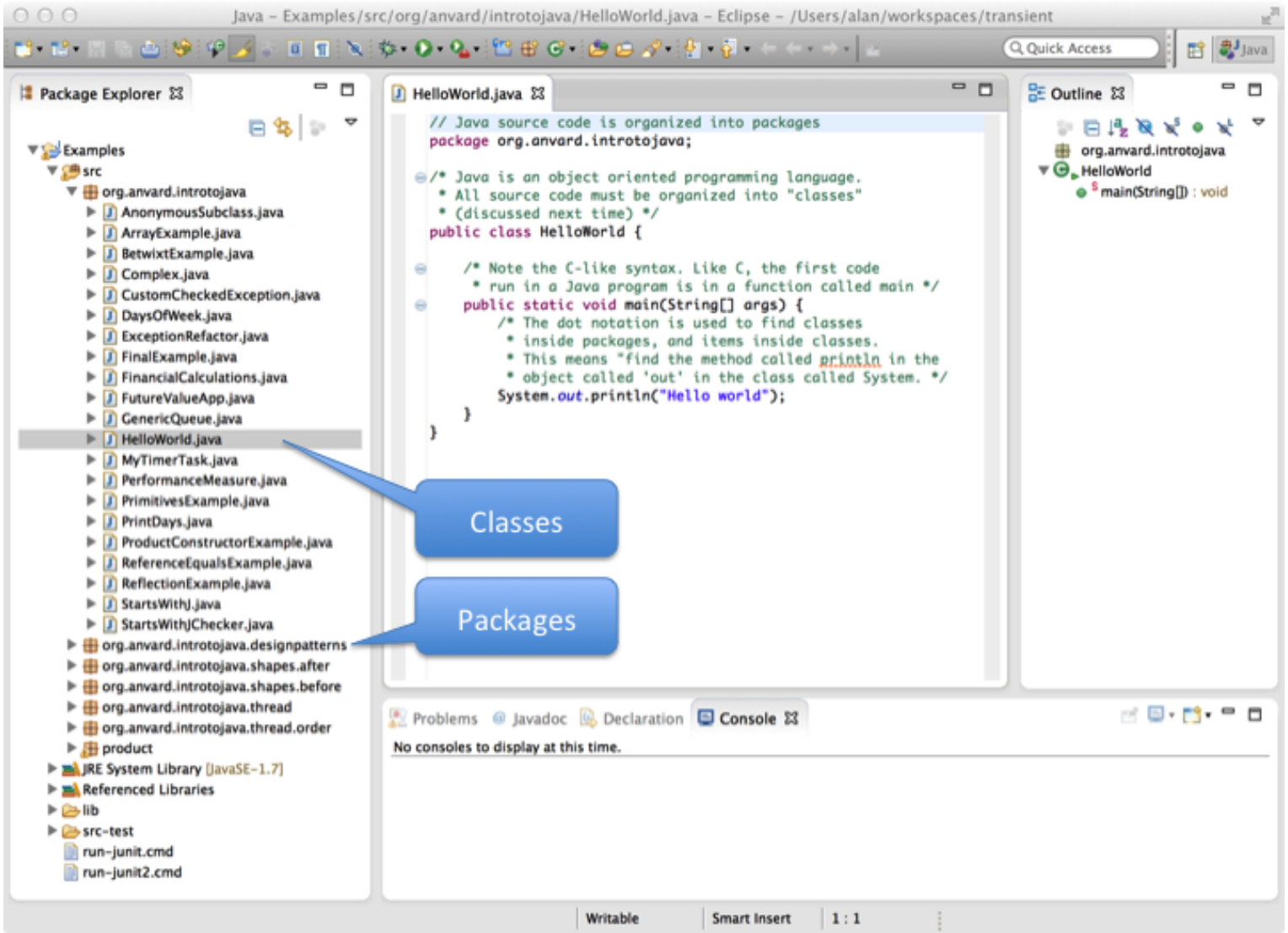
A Little More Syntax

```
package org.anvard.introtojava;

// Classes in other packages that we need
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class HelloName {

  public static void main(String[] args)
      throws Exception {
    System.out.print("What is your name? ");
    // 1 statement, 3 objects (more next time)
    String name = new BufferedReader(
        new InputStreamReader(System.in)).readLine();
    if (name.length() > 10) {
      System.out.println("You have a long name.");
    }
    // Smart concatenation, but no operator overloading
    System.out.println("Hello, " + name);
  }
}
```

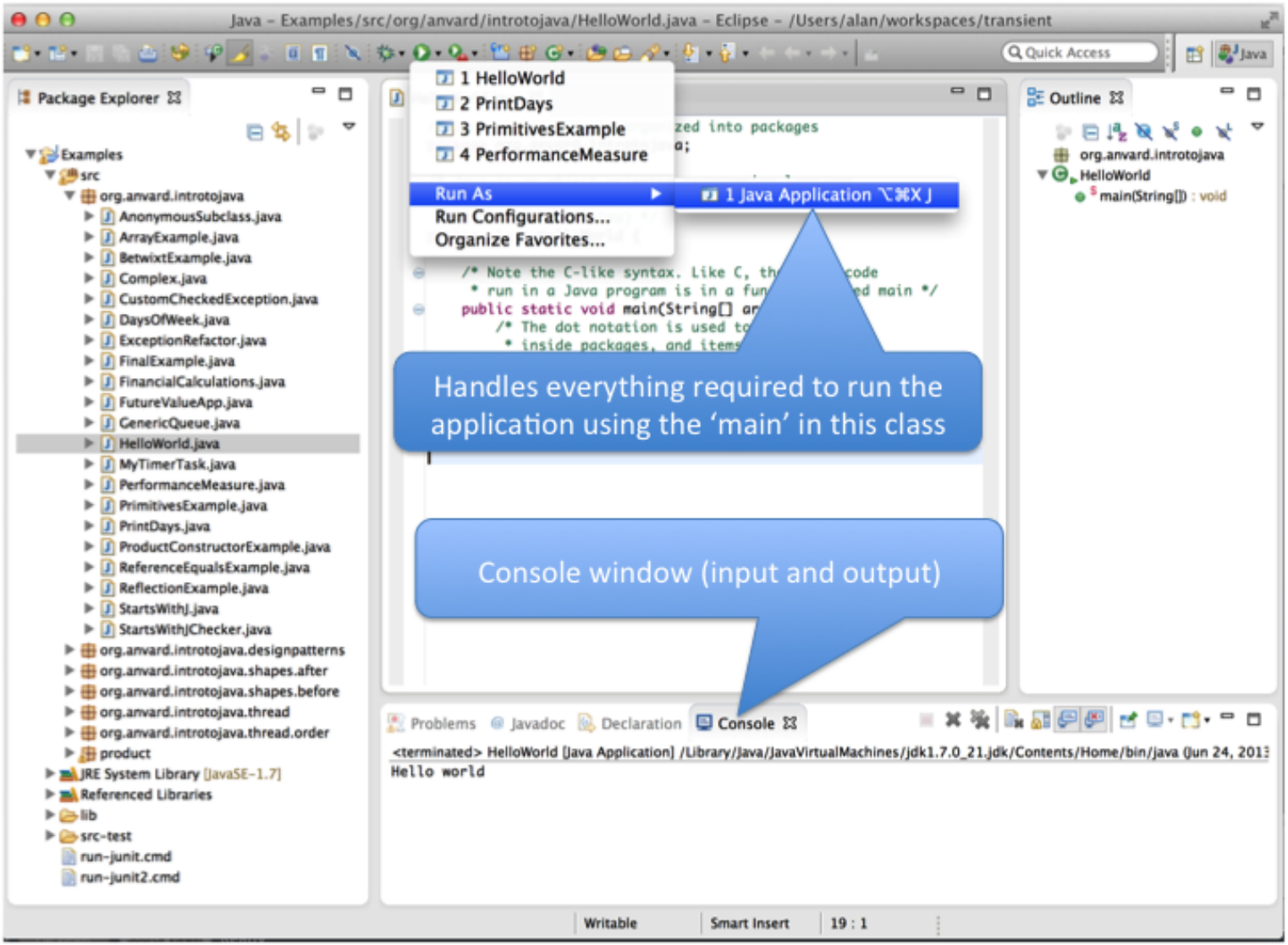
Java Memory Management

- That program raises questions about memory management
  - We used the `new` keyword to make some objects
  - We didn't "free" those objects, set them to `null`, or otherwise worry about them
- In Java, new objects are allocated on the "heap"
  - The JVM manages the heap
  - When the heap (or part of it) gets full, the JVM does "garbage collection"
  - This means identifying objects that are no longer referenced from live code and freeing the memory
- Object allocation happens all the time in Java
  - Most objects are short-lived
  - For example, the `readLine` method and the + operator both instantiated new string objects
  - Modern JVMs are optimized for lots of short-lived objects, so this is surprisingly performant
  - We'll talk later about how to avoid performance issues with object instantiation

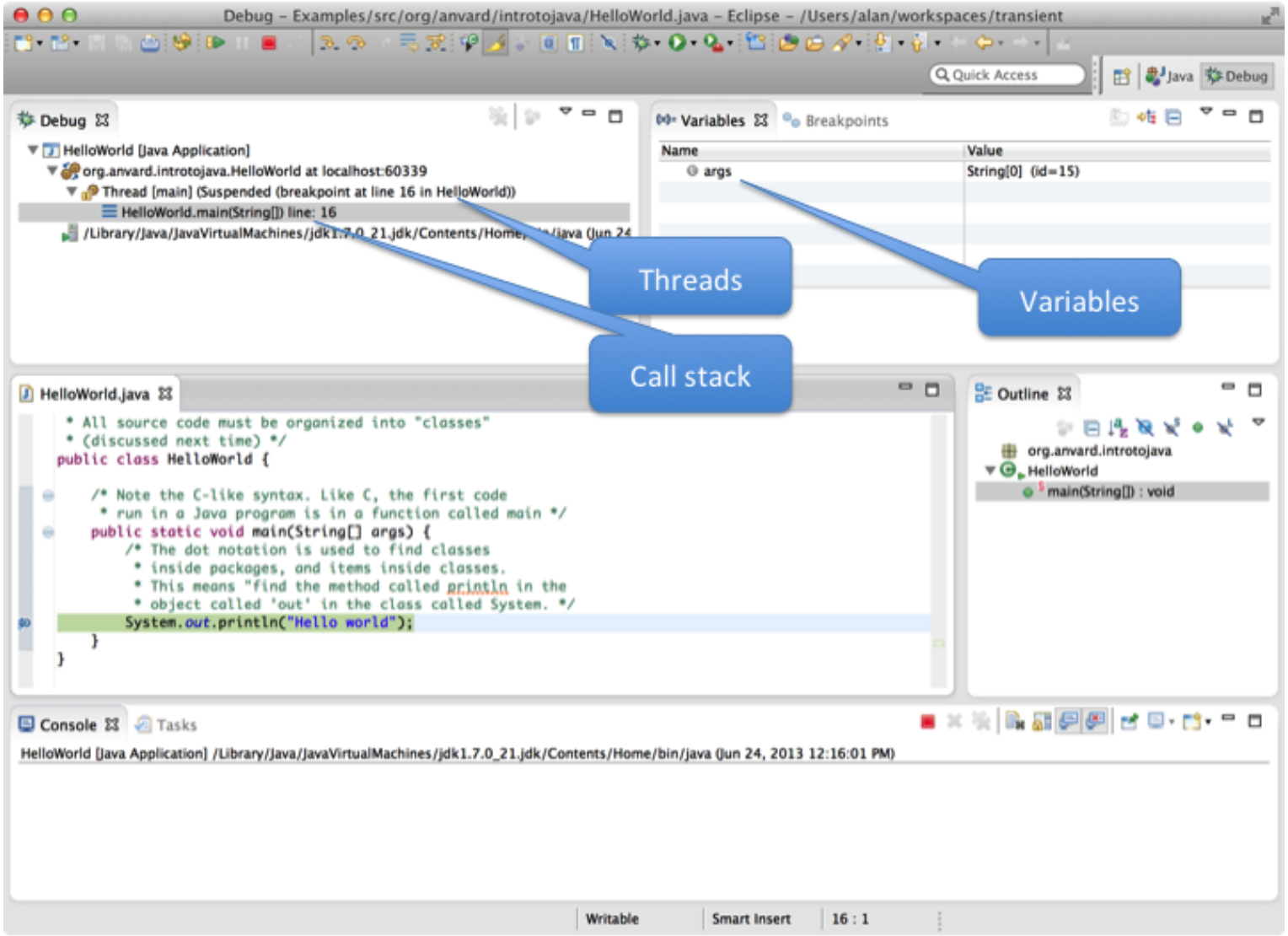
Editing in Eclipse



Running in Eclipse



Debugging in Eclipse



Working in Eclipse

- Eclipse performs continuous background compiling
  - Errors are displayed immediately
  - The compiled class files are kept in an "output" folder (`bin` by default)
- As dependencies are added to the "build path" for the project, they automatically become available for compiling and running
- Eclipse can also connect to externally running Java programs (locally or on a network) for debugging

Compiling from the command line

- The `javac` command is used for compiling from the command line
  - For all but simple examples, it is not usually used directly
  - Later we will discuss build tools such as Apache Ant
- Command line compiling must be done from the top level, not inside any packages
- Any required libraries must be specified in the "classpath"

This command creates a HelloWorld.class in the same directory as HelloWorld.java

```
$ javac org/anvard/introtojava/HelloWorld.java
```

Running from the command line

- Java programs can be run directly using the command `java`
- Later we will talk about packaging applications to avoid the command line
- When running a Java program, we specify the "main class", so it is OK for multiple Java classes to have a "main" method

```
$ java -cp . org.anvard.introtojava.HelloWorld
Hello world

$ java -cp . org.anvard.introtojava.HelloName
What is your name? John Jacob Jingleheimer Schmidt
You have a long name.
Hello, John Jacob Jingleheimer Schmidt
```

The Java "Classpath"

- The classpath tells Java where to look for class files
  - For compiling, this means already compiled code used as a dependency
  - For running, this means all classes the Java program will need
- The "classpath" is just a list of locations to search for class files and other resources
- The individual items can be directories, archive files, or arbitrary URLs
- The standard Java library is always on the classpath
- For our simple example, we just needed the current directory on the classpath

```
$ java -cp . org.anvard.introtojava.HelloWorld
Hello world
```

Java Classpath issues

- Classpath issues can be a source of frustration in running Java programs
- Classes are found and loaded dynamically, so classpath issues might not be discovered until the program is running
- Multiple classes on the classpath can have the same name
  - The “first” one will be used
  - This can be confusing
  - Good package naming and organization will usually avoid this problem

Next Time

- Java Objects and Classes
- Brief overview of Object-Oriented Programming
- Distinction between primitive types and objects
- Distinction between static and instance variables

Credit in LMPeople

LMPeople Course Code: 071409ILT01