Intro to Java

A workshop

Java platform overview

Java technology is used to develop applications for a wide range of environments, from consumer devices to heterogeneous enterprise systems.

The Java language

- The Java language has its own structure, syntax rules, and programming paradigm.

- The Java language's programming paradigm is based on the concept of object-oriented programming (OOP), which the language's features support.

The Java language

The Java language is a C-language derivative, so its syntax rules look much like C's: for example,code blocks are modularized into methods and delimited by braces ({ and }), and variables are declared before they are used.

The Java language

- Structurally, the Java language starts with packages.
- A package is the Java language's namespace mechanism.
- Within packages are classes, and within classes are methods, variables, constants, and so on.

The Java compiler

When you program for the Java platform, you write source code in .java files and then compile them.

- The compiler checks your code against the language's syntax rules, then writes out bytecodes in .class files.
- Bytecodes are standard instructions targeted to run on a Java virtual machine (JVM).

The Java compiler

- In adding this level of abstraction, the Java compiler differs from other language compilers, which write out instructions suitable for the CPU chipset the program will run on.

The JVM

- At run time, the JVM reads and interprets .class files and executes the program's instructions on the native hardware platform for which the JVM was written.
- The JVM interprets the bytecodes just as a CPU would interpret assembly-language instructions. The difference is that the JVM is a piece of software written specifically for a particular platform.
- The JVM is the heart of the Java language's "write-once, run-anywhere" principle.

The JVM

- Your code can run on any chipset for which a suitable JVM implementation is available.

- JVMs are available for major platforms like Linux and Windows, and subsets of the Java language have been implemented in JVMs for mobile phones and hobbyist chips.

The garbage collector

- Rather than forcing you to keep up with memory allocation (or use a third-party library to do this),
- the Java platform provides memory management out of the box.
- When your Java application creates an object instance at run time, the JVM automatically allocates memory space for that object from the heap, which is a pool of memory set aside for your program to use.

The garbage collector

- The Java garbage collector runs in the background, keeping track of which objects the application no longer needs and reclaiming memory from them.
- This approach to memory handling is called implicit memory management because it doesn't require you to write any memory-handling code.
- Garbage collection is one of the essential features of Java platform performance.

The Java Development Kit

- When you download a Java Development Kit (JDK), you get — in addition to the compiler and other tools — a complete class library of prebuilt utilities that help you accomplish just about any task common to application development.
- The best way to get an idea of the scope of the JDK packages and libraries is to check out the JDK API documentation

The Java Runtime Environment

- The Java Runtime Environment (JRE) includes the JVM, code libraries, and components that are necessary for running programs written in the Java language.
- It is available for multiple platforms.
- You can freely redistribute the JRE with your applications, according to the terms of the JRE license, to give the application's users a platform on which to run your software.
- The JRE is included in the JDK

Your development environment

- The JDK includes a set of command-line tools for compiling and running your Java code, including a complete copy of the JRE.

- Although you certainly can use these tools to develop your applications, most developers appreciate the additional functionality, task management, and visual interface of an IDE.

Your development environment

- Eclipse is a popular open source IDE for Java development.
- It handles basic tasks, such as code compilation and setting up a debugging environment, so that you can focus on writing and testing code.
- In addition, you can use Eclipse to organize source code files into projects, compile and test those projects, and store project files in any number of source repositories.
- You need an installed JDK in order to use Eclipse for

Install JDK

- go to http://www.oracle.com/technetwork/java/javase/downloads/index.html
- download Java SE 7u51 JDK
- run installer, should install to C:\Program Files\Java\jdk1.7.0_51
- create JAVA_HOME variable

Install Eclipse

- go to
  - https://www.eclipse.org/downloads
- download Eclipse Standard 4.3.2
- unzip and move to C:\eclipse

Setup Eclipse

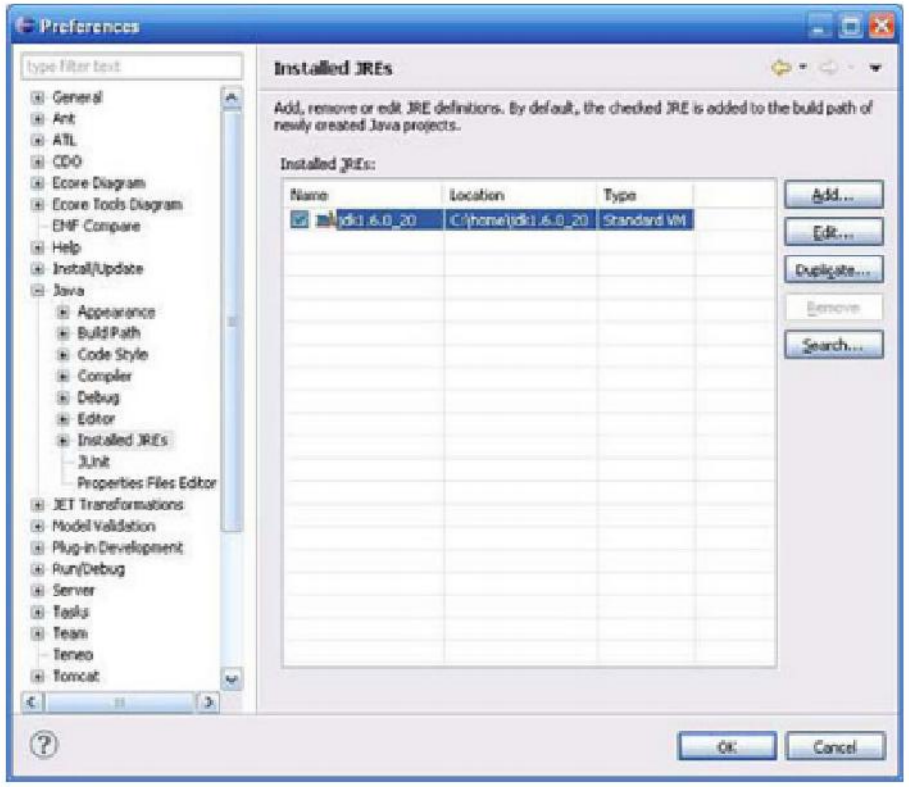
- The Eclipse IDE sits atop the JDK as a useful abstraction, but it still needs to access the JDK and its various tools.
- Before you can use Eclipse to write Java code, you have to tell it where the JDK is located.

Setup Eclipse

- Launch Eclipse by double-clicking on eclipse.exe
- The Workspace Launcher will appear, allowing you to select a root folder for your Eclipse projects.
- Choose a folder you will easily remember, such as C:\workspace.
- Dismiss the Welcome to Eclipse screen.

Setup Eclipse

Click Window > Preferences > Java > Installed JREs.



Setup Eclipse

- Eclipse will point to an installed JRE.
- You need to make sure you use the one you downloaded with JDK.
- If Eclipse does not automatically detect the JDK you installed, click Add... and in the next dialog Standard VM, then click Next.
- Specify the JDK's home directory (such as C:\Program Files\Java\jdk1.7.0_51 on Windows), then click Finish.
- Confirm that the JDK you want to use is selected and

Eclipse setup complete

Eclipse is now set up and ready for you to create projects and compile and run Java code.

Getting started with Eclipse

- Eclipse is not just an IDE, it is an entire development ecosystem.
- This section is a brief hands-on introduction to using Eclipse for Java development.

Eclipse development environment

The Eclipse development environment has four main components:

- Workspace
- Projects
- Perspectives
- Views

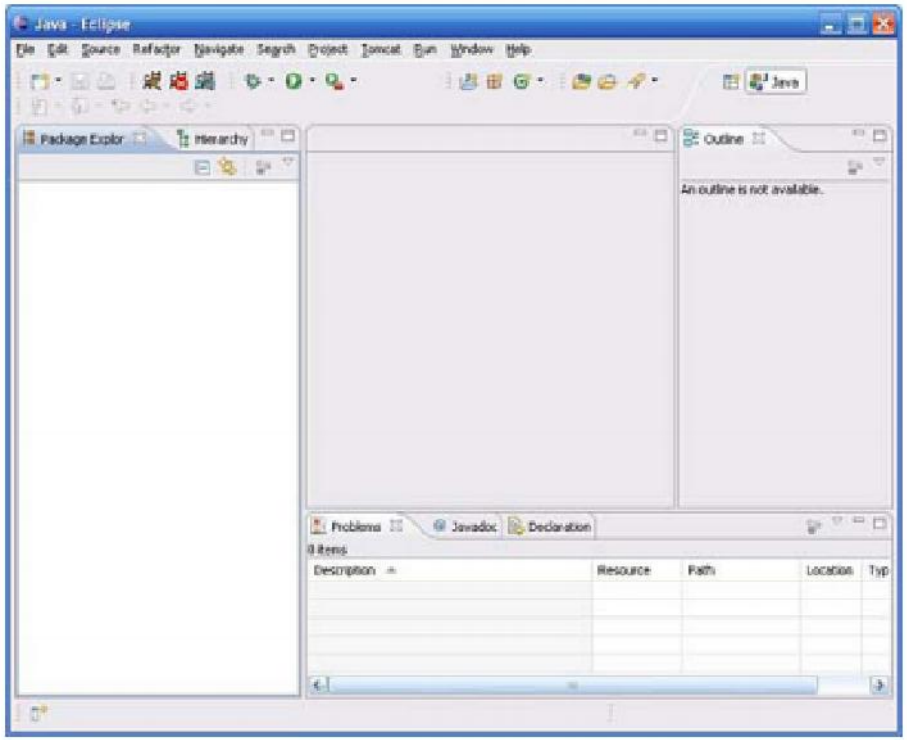
Eclipse development environment

- The primary unit of organization in Eclipse is the workspace.
- A workspace contains all of your projects.
- A perspective is a way of looking at each project (hence the name), and within a perspective are one or more views.

The Java perspective

- Is the default perspective for Eclipse
- You should see this perspective when you start up Eclipse
- The Java perspective contains the tools you need to begin writing Java applications.
- Each tab is a view for the Java perspective.
- Package Explorer and Outline are two particularly useful views.

The Java perspective



The Java perspective

- The Eclipse environment is highly configurable.
- Each view is dockable, so you can move it around in the Java perspective and place it where you want it.
- For now, though, stick with the default perspective and view setup.

What is an object?

- Structured programming languages like C and COBOL follow a very different programming paradigm from object-oriented ones
- The structured programming paradigm is highly data oriented, which means that you have data structures on one hand, and then program instructions that act on that data.
- Object-oriented languages like the Java language combine data and program instructions into *objects*.

What is an object?

- An object is a self-contained entity that contains attributes and behavior, and nothing more.

- Rather than having a data structure with fields (attributes) and passing that structure around to all of the program logic that acts on it (behavior), in an object-oriented language, data and program logic are combined.

- This combination can occur at vastly different levels of granularity, from fine-grained objects like a “Number” to coarse-grained objects such as a “FundsTransfer” service in a large banking application.

Object definition

```
1  package packageName;
2  import ClassNameToImport;
3  accessSpecifier class ClassName {
4      accessSpecifier dataType variableName [= initialValue];
5      accessSpecifier ClassName([argumentList]) {
6          constructorStatement(s)
7      }
8      accessSpecifier returnType methodName([argumentList]) {
9          methodStatement(s)
10     }
11 // This is a comment
12 /* This is a comment too */
13 /* This is a
14 multiline
15 comment */
16 }
17
```

Note: square brackets indicate that the constructs within them are not required. The brackets themselves (unlike { and }) are not part of the Java syntax.

Class makeup

- Classes can have two types of members:
  - variables
  - methods

Variables

- The values of a given class's variables distinguish each instance of that class and define its state.
- These values are often referred to as instance variables

Methods

- A class's methods define its behavior.
- Sometimes this behavior is nothing more than to return the current value of an attribute.
- Other times, the behavior can be quite complex.

Methods

- There are essentially two categories of methods: constructors and all other methods, of which there are many types.
- A constructor method is used only to create an instance of a class.
- Other types of methods can be used for virtually any application behavior.

Static methods

- Static methods are used largely for utility; you can think of them as a way of having global methods.
- For example, you use the JDK Logger class to output information to the console.
- To create a Logger class instance, you don't instantiate a Logger class; instead, you invoke a static method called getLogger().
- The syntax for invoking a static method is different from the syntax used to invoke a method on an object instance. You also use the name of the class that contains the static method, as shown in this invocation:
    - Logger log = Logger.getLogger("NewLogger");
- So to invoke a static method, you don't need an object instance, just the name of the class

The Person Object

- I'll start with an example that is based on a common application-development scenario: an individual being represented by a Person object.
- Going back to the definition of an object, you know that an object has two primary elements:
  - attributes and behavior

Attributes

What attributes can a person have? Some common ones include:

- Name
- Age
- Height
- Weight
- Eye Color
- Gender

Basic class definition for *Person*

```
1  package com.orasi.intro;
2
3  public class Person {
4      private String name;
5      private int age;
6      private int height;
7      private int weight;
8      private String eyeColor;
9      private String gender;
10 }
```

- The basic class definition for Person isn't very useful at this point because it defines only its attributes (and private ones at that).
- To be more interesting, the Person class needs behavior — and that means methods.

Behavior

- An actual person can do all sorts of things, but object behaviors usually relate to some kind of application context.
- In a business-application context, for instance, you might want to ask your Person object, "What is your age?"
- In response, Person would tell you the value of its Age attribute.

Behavior

More-complex logic could be hidden inside of the Person object, but for now suppose that Person has the behavior of answering these questions:

- What is your name?
- What is your age?
- What is your height?
- What is your weight?

Person class with constructor

```java
package com.orasi.intro;

public class Person {
    private String name;
    private int age;
    private int height;
    private int weight;
    private String eyeColor;
    private String gender;
    public Person() {
        // Nothing to do...
    }

    public Person(String name, int age, int height, String eyeColor, String gender) {
        this.name = name;
        this.age = age;
        this.height = height;
        this.weight = weight;
        this.eyeColor = eyeColor;
        this.gender = gender;
    }
}
```

Your first Java object

It's time to pull together what you've learned in the previous sections and start writing some code.

- This section walks you through declaring a class and adding variables and methods to it using the Eclipse Package Explorer.
- You'll learn how to use the Logger class to keep an eye on your application's behavior.
- Also how to use a main() method as a test harness.

Create a project

Click on File > New > Java Project ... and you will see a dialog box open

New Java Project Wizard

Create a project

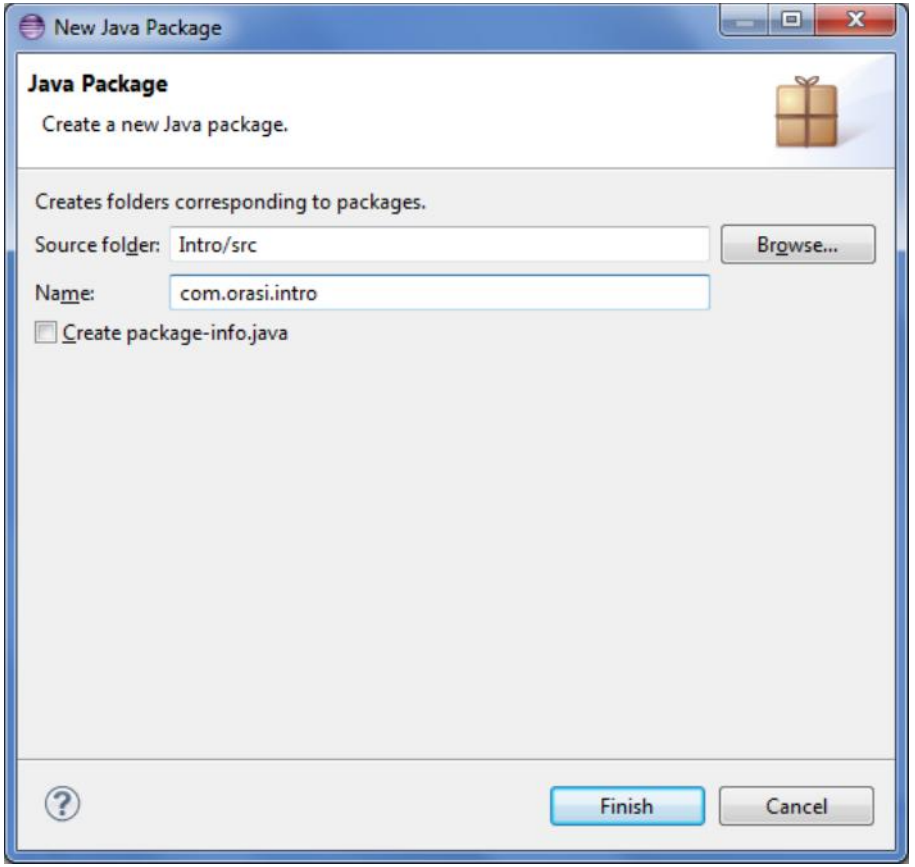
- Enter Intro as the project name and click Finish.
- If you want to modify the default project settings, click Next. (This is recommended only if you have experience with the Eclipse IDE.)
- Click Finish to accept the project setup and create the project

Creating a package

- If you're not already there, get to the Package Explorer perspective in Eclipse.
- You're going to get set up to create your first Java class.
- The first step is to create a place for the class to live.
- Packages are namespace constructs, but they conveniently map directly to the file system's directory structure as well.
- Rather than use the default package (almost always a bad idea), you'll create one specifically for the code

Eclipse Java package wizard

- File > New > Package
- Type com.orasi.intro
- Click Finish



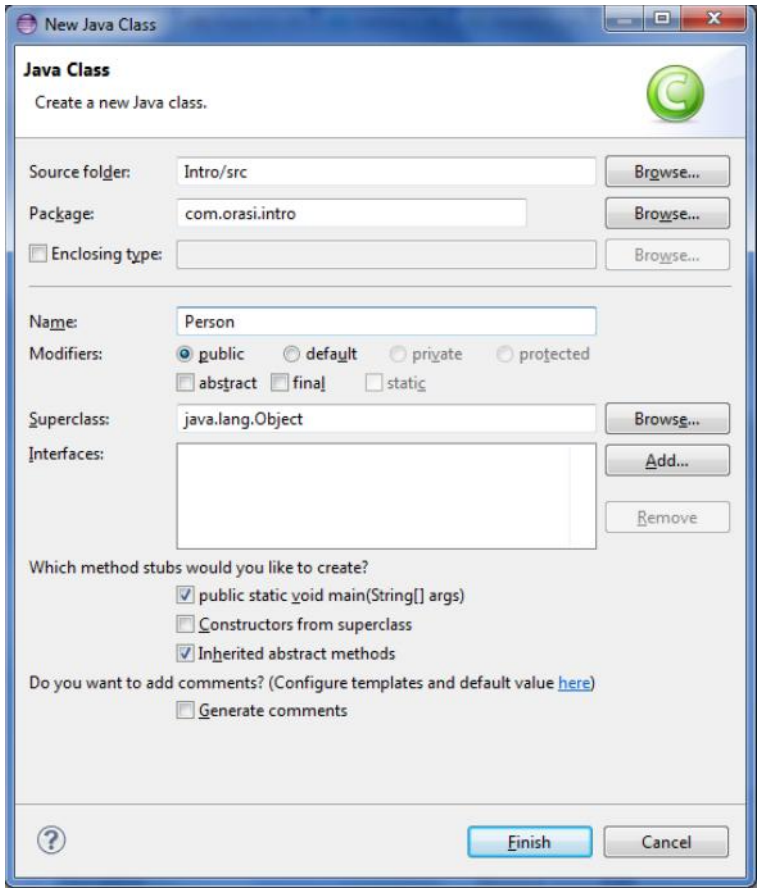
Declaring the class

There's more than one way to create a class from the Package Explorer, but the easiest way is to right-click on the package you just created and choose New > Class.... You will see the New Class dialog box.

New Class dialogue box

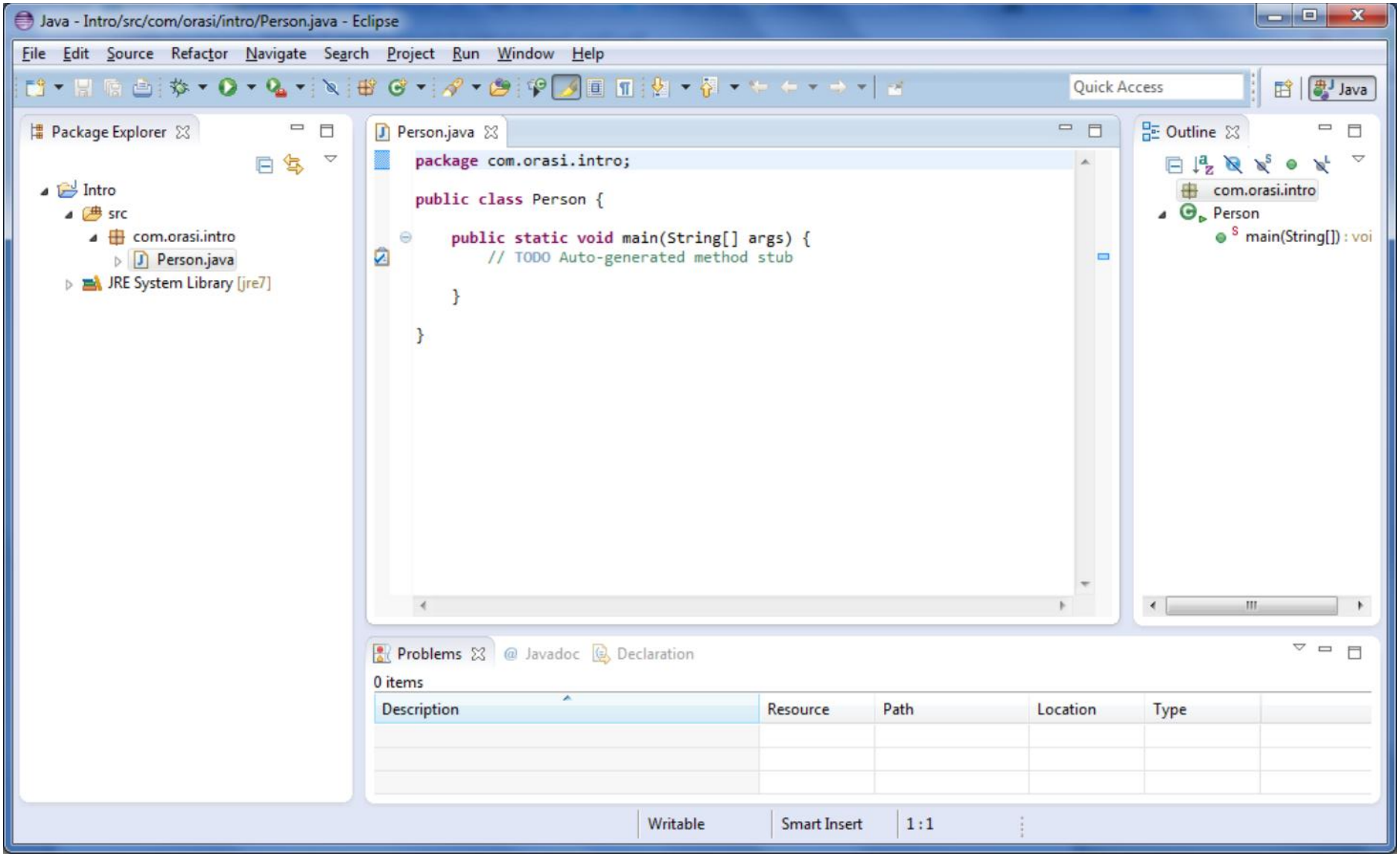
- In the Name text box, type Person.
- Under Which method stubs would you like to create?, check public static void main(String[] args).
- Next, click Finish.

New Class dialogue box



New Class

- Eclipse generates a shell class for you and includes the package statement at the top, along with the main() method you asked for and the comments you see.
- You just need to flesh out the class now. You can configure how Eclipse generates new classes via:
  - Window > Preferences > Java > Code Style > Code Templates
- For simplicity, you'll go with Eclipse's out-of-the-box code generation.


Java - Intro/src/com/orasi/intro/Person.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
Intro
src
com.orasi.intro
Person.java
JRE System Library [jre7]
Person.java
package com.orasi.intro;
public class Person {
public static void main(String[] args) {
// TODO Auto-generated method stub
}
}
Outline
com.orasi.intro
Person
main(String[]) : voi
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type
Writable
Smart Insert
1 : 1

Adding class variables

- Recall that a variable has an accessSpecifier, a dataType, a variableName, and, optionally, an initialValue.
- Earlier, you looked briefly at how to define the accessSpecifier and variableName.
- Now you'll see the dataType that a variable can have.

dataTypes

- A dataType can be either a primitive type or a reference to another object.
- For example, remember that Age is an int (a primitive type), and Name is a String (an object).
- The JDK comes packed full of useful classes like java.lang.String, and those in the java.lang package do not need to be imported (a shorthand courtesy of the Java compiler).
- But whether the dataType is a JDK class such as String or a user-defined class, the syntax is essentially the same.

Primitive data types

This table shows the eight primitive data types you're likely to see on a regular basis, including the default values that primitives take on if you do not explicitly initialize a member variable's value:

Type	Size	Default value	Range of values
`boolean`	n/a	false	true or false
`byte`	8 bits	0	-128 to 127
`char`	16 bits	(unsigned)	\u0000' \u0000' to \uffff' or 0 to 65535
`short`	16 bits	0	-32768 to 32767
`int`	32 bits	0	-2147483648 to 2147483647
`long`	64 bits	0	-9223372036854775808 to 9223372036854775807
`float`	32 bits	0.0	1.17549435e-38 to 3.4028235e +38
`double`	64 bits	0.0	4.9e-324 to 1.7976931348623157e+308

Built-in logging

- Before going further into coding, you need to know how your programs tell you what they are doing.
- The Java platform includes the java.util.logging package, a built-in logging mechanism for gathering program information in a readable form.
- Loggers are named entities that you create through a static method call to the Logger class, like so:
  - import java.util.logging.Logger;
  - //. . .
  - Logger log = Logger.getLogger(getClass().getName());

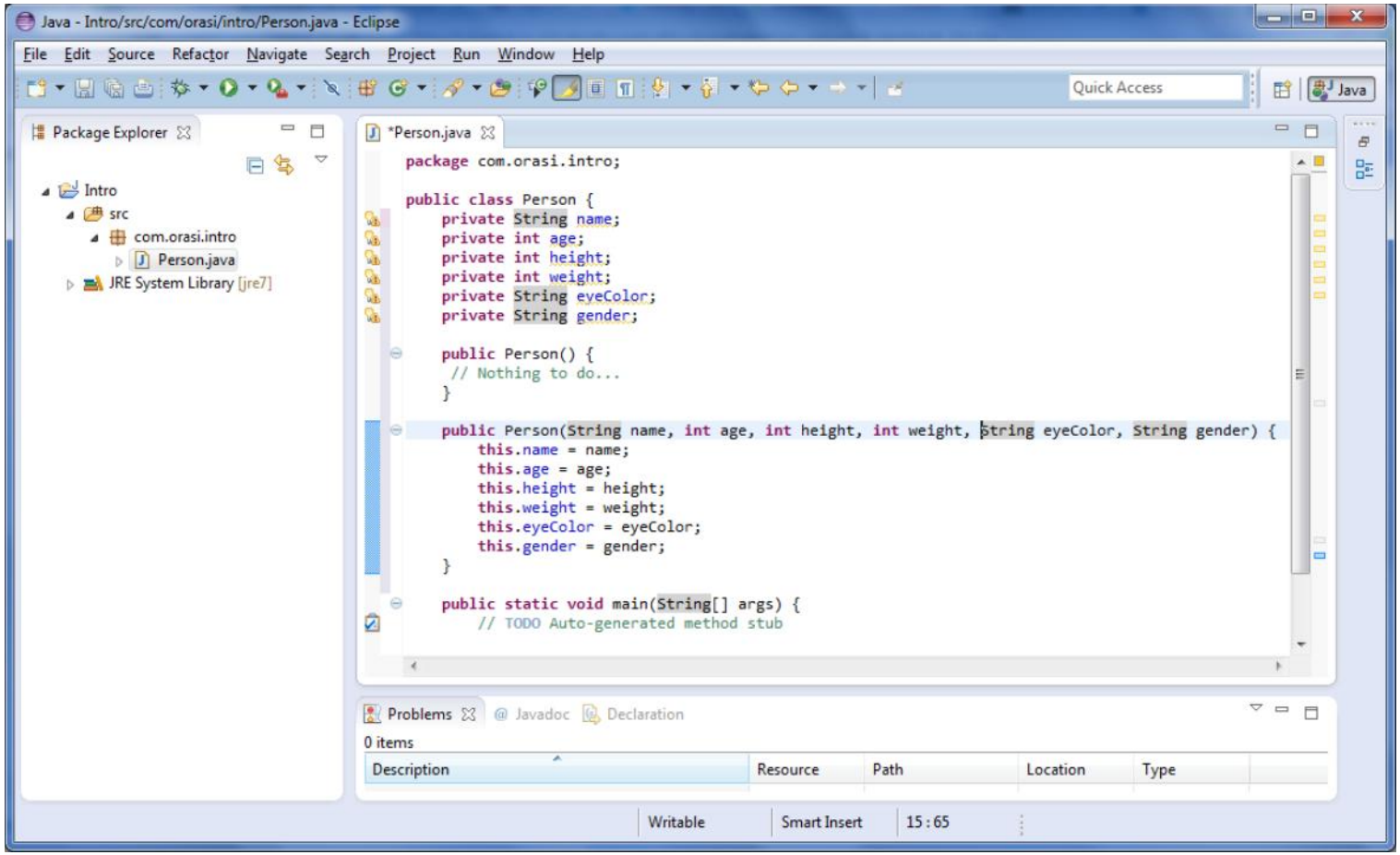
Built-in logging

- From any regular (that is, nonstatic) method, the previous code will always reference the name of the class and pass that to the Logger.
- If you are making a Logger call inside of a static method, just reference the name of the class you're inside of:
  - Logger log = Logger.getLogger(Person.class.getName());

Using main() as a test harness

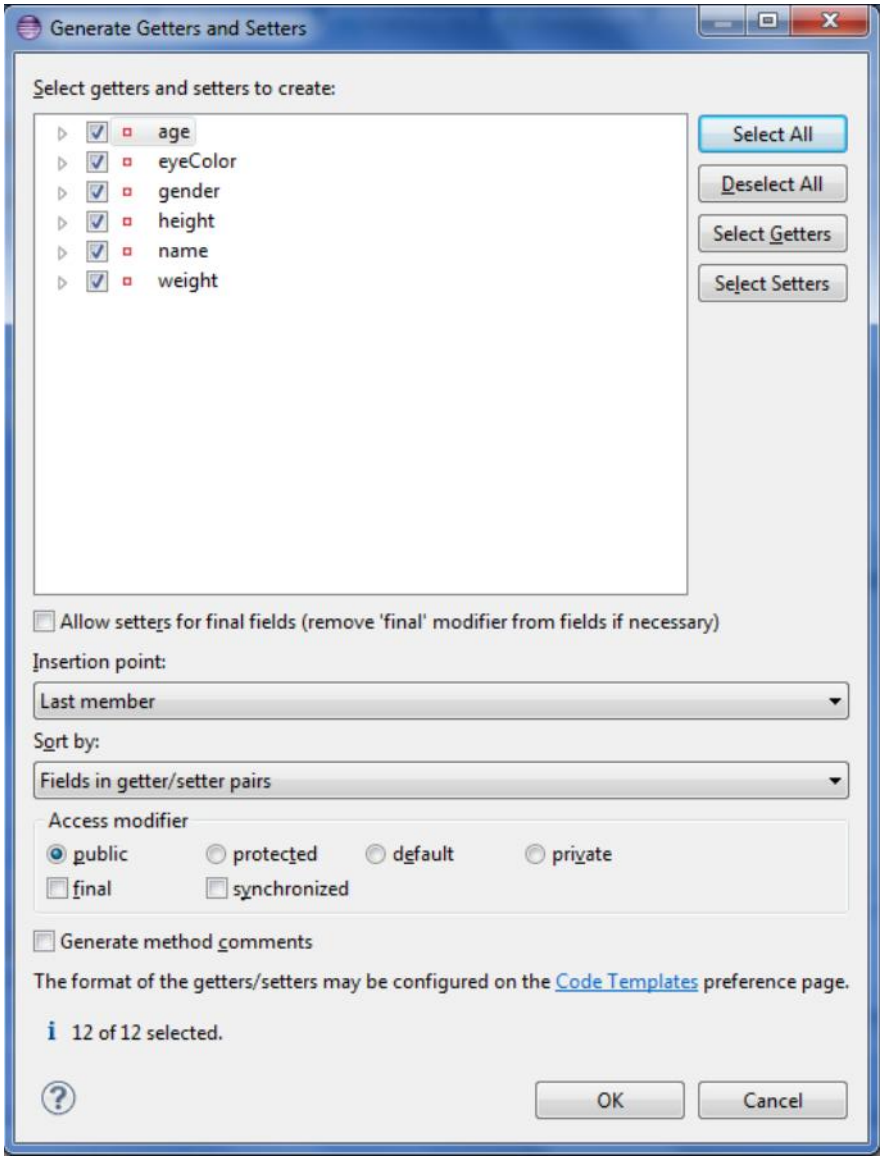
- main() is a special method that you can include in any class so that the JRE can execute its code.
- A class is not required to have a main() method, in fact, most never will.
- A class can have at most one main() method.

Add our attributes and constructors



Generate getters/setters

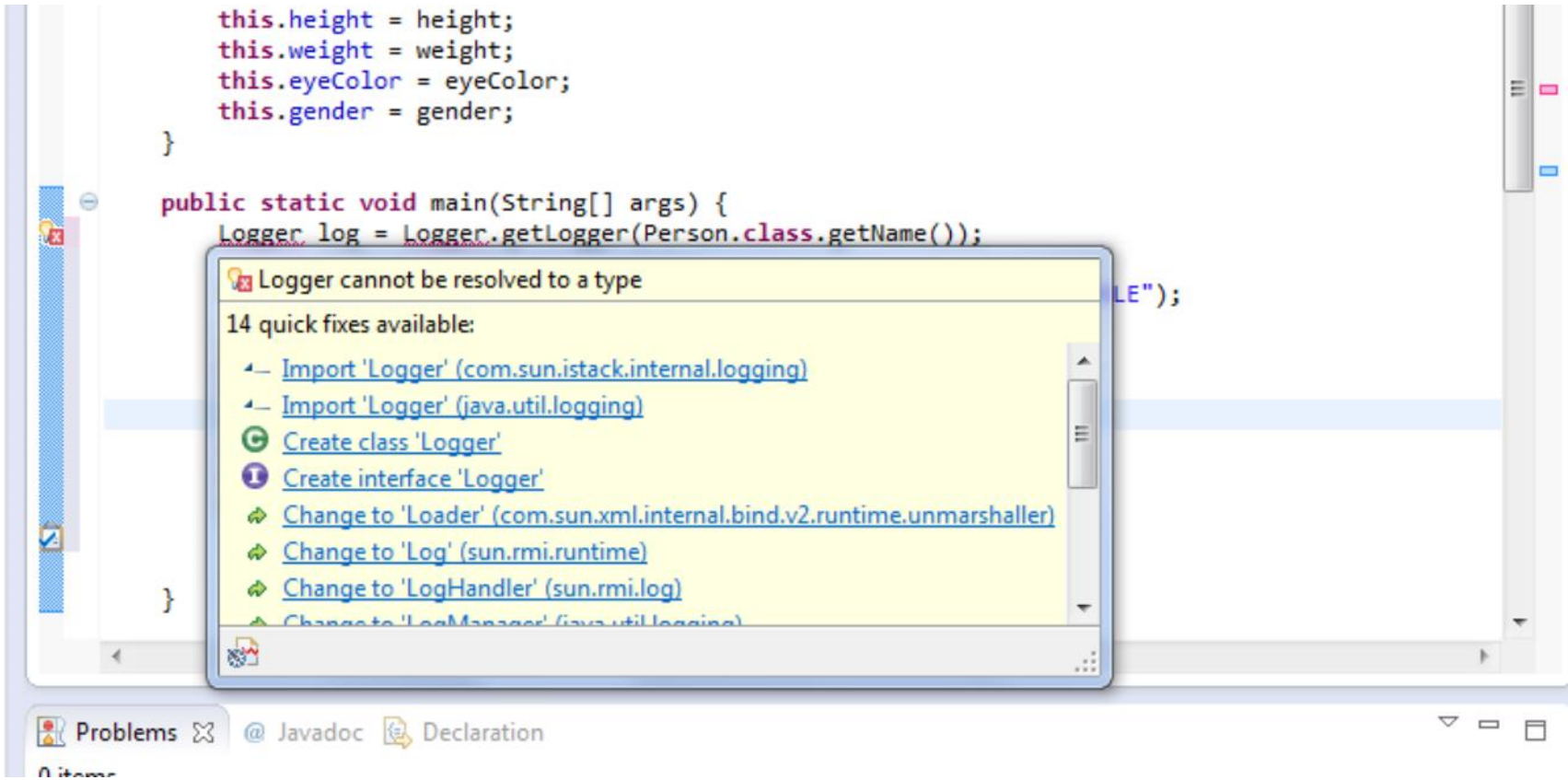
- Eclipse has a handy code generator to generate getters and setters (among other things).
  - Right click on Person.java or Person in “public class Person”
  - Source > Generate Getters and Setters....
  - Select All
  - Insertion point: Last Member
  - Click OK


Generate Getters and Setters
Select getters and setters to create:
age
eyeColor
gender
height
name
weight
Select All
Deselect All
Select Getters
Select Setters
Allow setters for final fields (remove 'final' modifier from fields if necessary)
Insertion point:
Last member
Sort by:
Fields in getter/setter pairs
Access modifier
public
protected
default
private
final
synchronized
Generate method comments
The format of the getters/setters may be configured on the Code Templates preference page.
12 of 12 selected.
OK
Cancel

Fill out main method

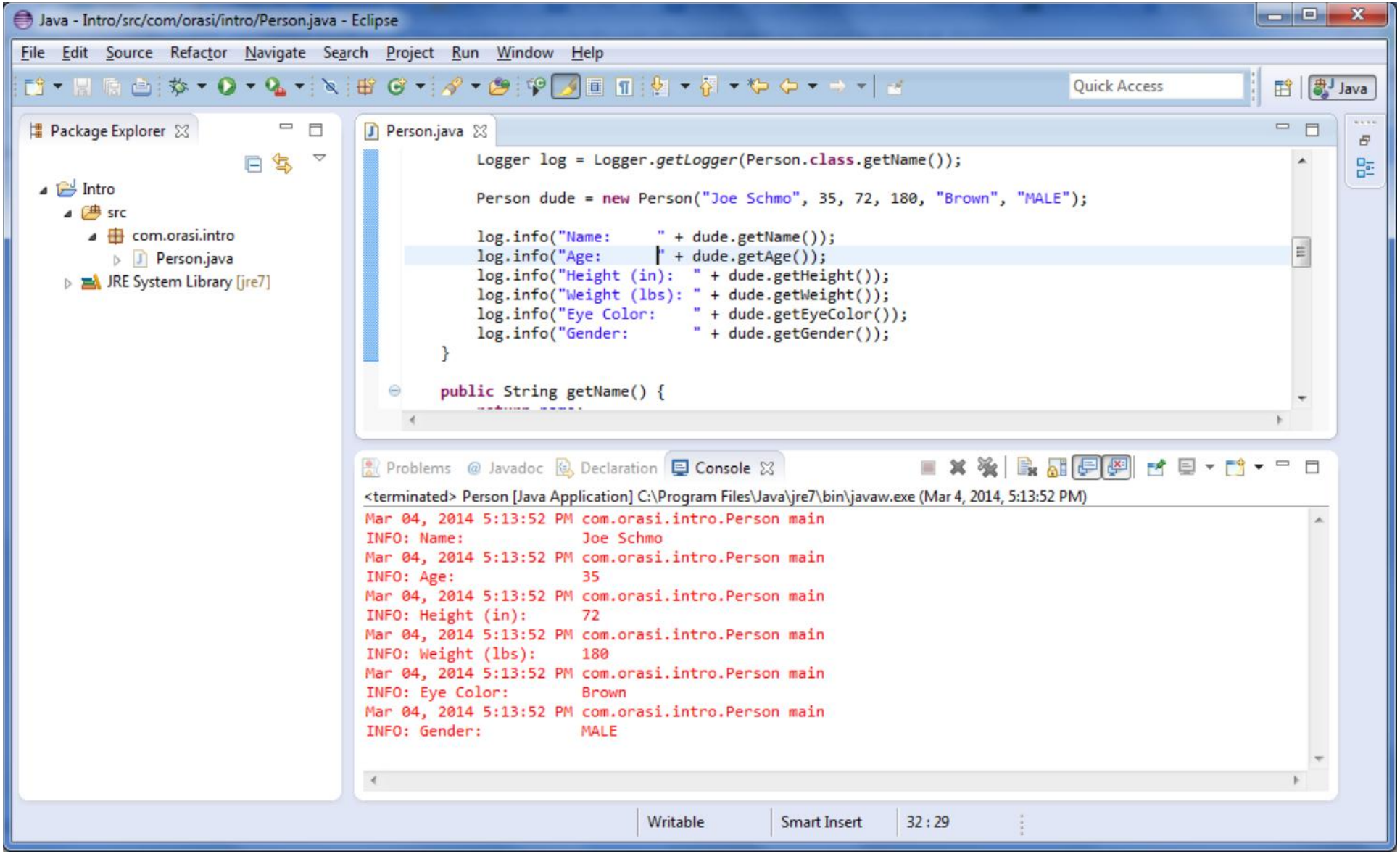
```
Logger log = Logger.getLogger(Person.class.getName());
Person dude = new Person("Joe Schmo", 35, 72, 180, "Brown", "MALE");
log.info("Name:                " + dude.getName());
log.info("Age:                 " + dude.getAge());
log.info("Height (in):         " + dude.getHeight());
log.info("Weight (lbs):        " + dude.getWeight());
log.info("Eye Color:           " + dude.getEyeColor());
log.info("Gender:              " + dude.getGender());
```

Add logger import



Executing code in Eclipse

- To run a Java application from inside Eclipse, select the class you want to run, which must have a main() method.
- Select Person, then click the Run icon (which is green and has a small triangular arrow pointing to the right).
- When asked, select to run Person as a Java application


Java - Intro/src/com/orasi/intro/Person.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
Intro
src
com.orasi.intro
Person.java
JRE System Library [jre7]
Person.java
Logger log = Logger.getLogger(Person.class.getName());
Person dude = new Person("Joe Schmo", 35, 72, 180, "Brown", "MALE");
log.info("Name: " + dude.getName());
log.info("Age: " + dude.getAge());
log.info("Height (in): " + dude.getHeight());
log.info("Weight (lbs): " + dude.getWeight());
log.info("Eye Color: " + dude.getEyeColor());
log.info("Gender: " + dude.getGender());
}
public String getName() {
Problems Javadoc Declaration Console
<terminated> Person [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 4, 2014, 5:13:52 PM)
Mar 04, 2014 5:13:52 PM com.orasi.intro.Person main
INFO: Name: Joe Schmo
Mar 04, 2014 5:13:52 PM com.orasi.intro.Person main
INFO: Age: 35
Mar 04, 2014 5:13:52 PM com.orasi.intro.Person main
INFO: Height (in): 72
Mar 04, 2014 5:13:52 PM com.orasi.intro.Person main
INFO: Weight (lbs): 180
Mar 04, 2014 5:13:52 PM com.orasi.intro.Person main
INFO: Eye Color: Brown
Mar 04, 2014 5:13:52 PM com.orasi.intro.Person main
INFO: Gender: MALE
Writable
Smart Insert
32 : 29

Strings

- Handling strings in C is labor-intensive because they're null-terminated arrays of 8-bit characters that you have to manipulate.
- In the Java language, strings are first-class objects of type String, with methods that help you manipulate them.
- The closest Java code gets to the C world with regard to strings is the char primitive data type, which can hold a single Unicode character, such as ‘a’.

String instantiation

- Because Strings are first-class objects in the Java language, you can use new to instantiate them.
  - greeting = new String("hello")
- Setting a variable of type String has the same result, because the Java language creates a String object to hold the literal, then assigns that object to the instance variable.
  - String greeting = "hello";

Concatenate strings

```
log.info("Name: " + dude.getName());
```

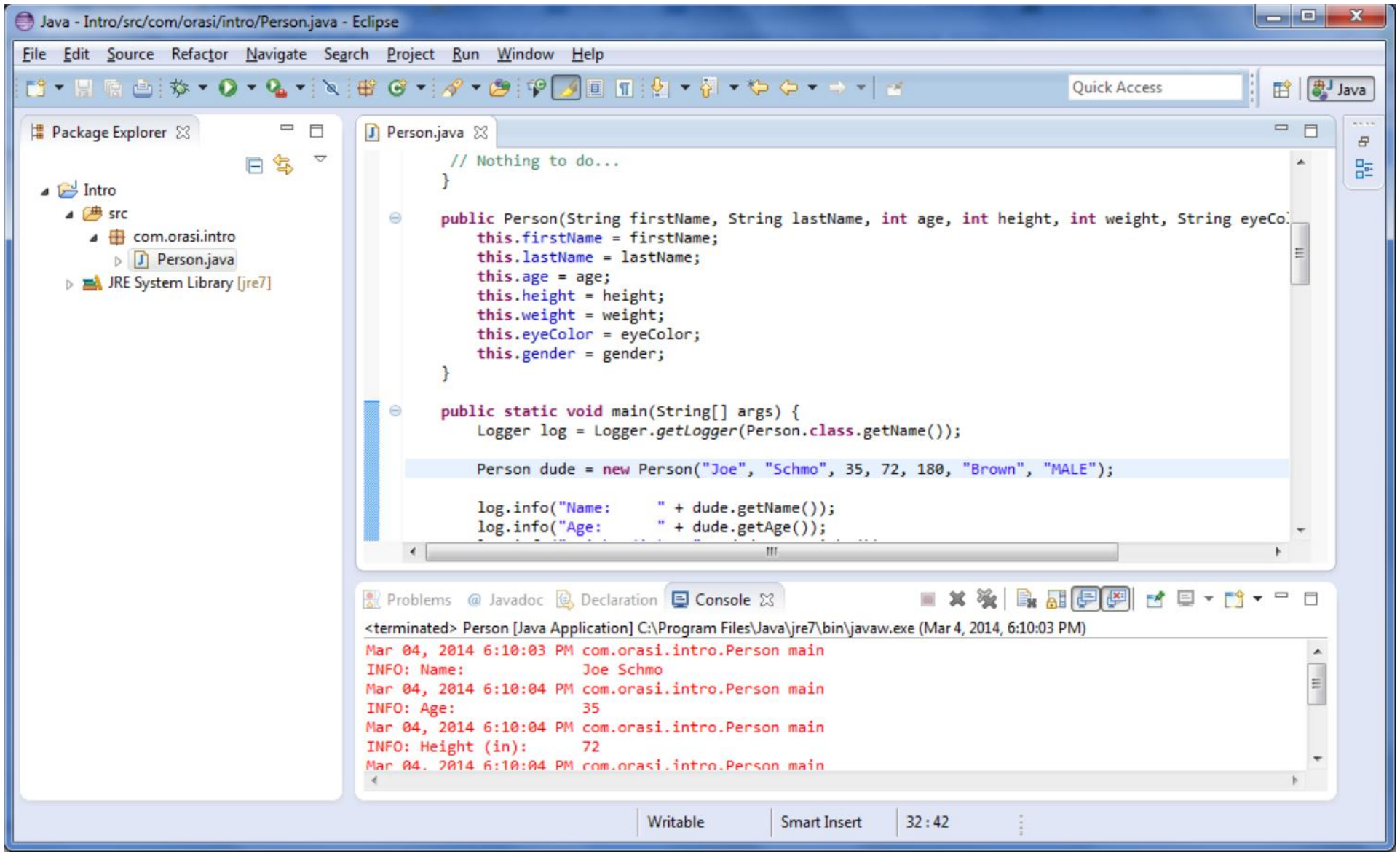
The plus (+) sign is shorthand for concatenating Strings in the Java language.

Concatenate example

- Let's try concatenating Strings inside of the Person class.
- At this point, you have a name instance variable, but it would be nice to have a firstName and lastName.
- You could then concatenate them when another object requests Person's full name.
- The first thing you need to do is add the new instance variables (at the same location in the source code where name is currently defined):
  - //private String name;
  - private String firstName;
  - private String lastName;

Chaining method calls

- Now you can generate getters and setters for firstName and lastName like we did before.
- Remove the setName() method, and change getName() to look like this:
  - public String getName() {
  - return firstName.concat(" ").concat(lastName);
  - }
- This code illustrates chaining of method calls


Java - Intro/src/com/orasi/intro/Person.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
Intro
src
com.orasi.intro
Person.java
JRE System Library [jre7]
Person.java
// Nothing to do...
}
public Person(String firstName, String lastName, int age, int height, int weight, String eyeCo
this.firstName = firstName;
this.lastName = lastName;
this.age = age;
this.height = height;
this.weight = weight;
this.eyeColor = eyeColor;
this.gender = gender;
}
public static void main(String[] args) {
Logger log = Logger.getLogger(Person.class.getName());
Person dude = new Person("Joe", "Schmo", 35, 72, 180, "Brown", "MALE");
log.info("Name: " + dude.getName());
log.info("Age: " + dude.getAge());
Problems Javadoc Declaration Console
<terminated> Person [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 4, 2014, 6:10:03 PM)
Mar 04, 2014 6:10:03 PM com.orasi.intro.Person main
INFO: Name: Joe Schmo
Mar 04, 2014 6:10:04 PM com.orasi.intro.Person main
INFO: Age: 35
Mar 04, 2014 6:10:04 PM com.orasi.intro.Person main
INFO: Height (in): 72
Writable
Smart Insert
32 : 42

Relational and conditional operators

- The Java language gives you operators and control statements that let you make decisions in your code.
- Most often, a decision in code starts with a boolean expression (that is, one that evaluates to either true or false).
- Such expressions use relational operators, which compare one operand or expression to another, and conditional operators.

The if/else statement

```
public int getHeight() {
        int ret = height;
    if (gender.equals("MALE")) {
            ret = height + 2;
        }
    else {
            ret = height;
            Logger.getLogger("Person").info("Being honest about height...");
    }
    return ret;
}
```

Archiving Java code

- The JDK ships with a tool called JAR, which stands for Java Archive.
- Creating a JAR file in Eclipse is a snap.
- Right-click the com.orasi.intro package and select Export.
- Choose Java > JAR file
- Browse to the location where you want to

External Archive

So how do you use someone elses 3rd party jar files?

- Right-click on project
- Select Build Path then Add External Archives…
- Browse to jar file

End

Configure proxy for Eclipse

- In Eclipse IDE, select “Window –> Preferences”
- Choose General -> Network Connections
- Select “Manual” from Action Provider drop down list
- Select Http in the List and click “Edit” button
- Fill in the proxy server host and port number, (fill in the username and password if any)

Install m2eclipse plugin

- Help -> Install New Software…
- Enter this url at Work with:
  - http://download.eclipse.org/technology/m2e/releases
- Hit Enter (not clicking Add)
- Expand, click Select All
- Click Next

Convert to Maven project

- Right-click on project in Package Explorer, select Copy
- Right-click in Package Explorer, select Paste
- Give name: Intro-Maven
- Right-click on new project in Package Explorer
- Select Configure -> Convert to Maven Project

Enter com.orasi for GroupId

Inspect POM file

Click on pom.xml tab

Note tags for src and test directories

Look in your workspace project directory and see where the pom.xml is located

Missing Dependencies?

- Click Dependencies tab on pom.xml
- Click Add
- Enter GAV: junit, junit, 4.11
- Select scope: test

JDK vs JRE

Maven requires JDK, so you may need to configure if using a JRE.

- Right-click project
- Build Path -> Configure Build Path…
- Select Libraries tab
- Select JRE System Library

Run

Click run icon (green triangle)

Select Clean

Click again select Install

Other Maven Eclipse config

Click Windows -> Preferences

Select Maven