MAHARISHI UNIVERSITY of MANAGEMENT

Engaging the Managing Intelligence of Nature

Computer Science Department

CS390 Foundamental Programming Practices (FPP)

Professor Paul Corazza

Lecture 1: Introduction to Java And the Eclipse Development Environment

Pulling the Arrow Back to Hit the Target

Wholeness of the Lesson

Java is an object-oriented highly portable programming language that arose as an easy alternative to the once dominant, but error-prone, C++ language. Eclipse is one of many open source, powerful but easy-to-use integrated development environments for use with Java and related technologies. Working from deeper levels of intelligence allows one to accomplish more with fewer mistakes and less effort.

About Java

- ***Brief History.*** The Java language began as a language for programming electronic devices, though the original project was never completed. Its creator was James Gosling, of Sun Microsystems. The language was developed privately starting in 1991, and was made publicly available in 1994. In 2009, Oracle bought the rights to Java from Sun Microsystems.

- ***Interpreted Language.*** When you "compile" Java code, the result is not executable binary code, targeted to a particular machine; instead, the result is **bytecode**, having a portable intermediate code format. The bytecode is then executed by running an **interpreter,** called the **Java Virtual Machine** (JVM). This approach makes Java code highly portable; Java will run on any platform for which a JVM has been created.

About Java

- ***JIT.*** Interpreters run much more slowly than native binary executable code. Java performance is boosted considerably in modern-day JVMs because of the inclusion of a **just-in-time compiler**; this feature compiles frequently occurring bytecode sequences into native binary code and caches the results, re-using the cache as needed. The result is that Java runs almost as fast as C++ in typical cases.
- ***No Pointers.*** Unlike C and C++, Java does not make use of pointers; developers are not required to manage the memory usage of their applications. Memory management is handled automatically in Java by means of a **garbage collector.** The garbage collector is run by the JVM at different times during program execution to return to memory all unusued object references.

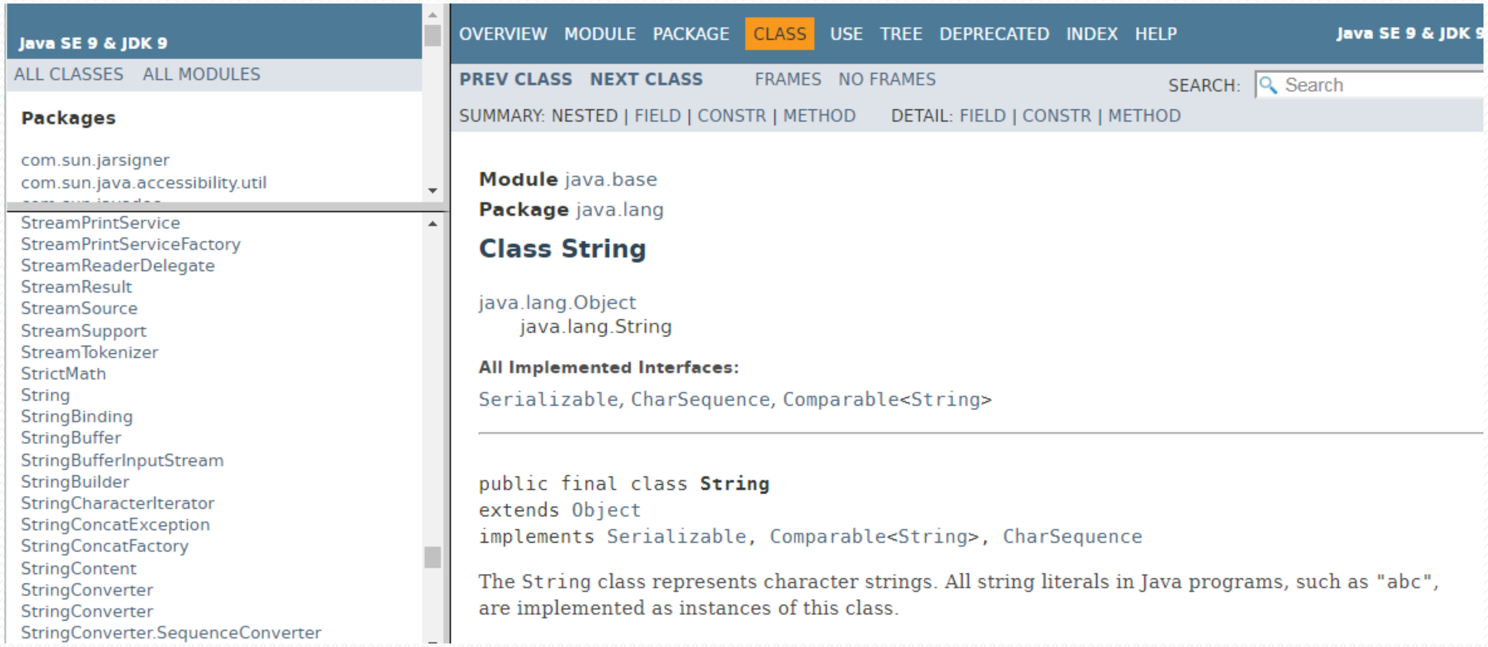
About Java

- ***Java Is an OOP Language.*** Java is an OO programming language, and succeeds in this better than C++, which (for the sake of backwards compatibility with C) supports a non-OO programming style (in C++, OO programming is "optional").
- ***Convenient Libraries.*** Java provides convenient libraries for handling files and streams, networking, http, gui development, and database connection. Compared to languages like C and C++, the increased ease of use is significant.
- ***Good Security Model.*** Java has a relatively good security model to support use over networks and distibuted environments. Though security holes are still found sometimes, these are rare and quickly patched. The fuss about *applet security holes* was exaggerated; applets are far safer than the Windows executables that are downloaded to (and that infect) Windows machines all the time without much discussion about the potential dangers.

The Java 9 API Docs

- Oracle provides online documentation of all the Java library classes. Full documentation of each class in the Java libraries is provided. For Java 9, the link is

  https://docs.oracle.com/javase/9/docs/api/index.html?overview-summary.html



Module java.base
Package java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

The JDK Commandline Tools

- To compile and run Java from a command window, you will use the executables javac.exe and java.exe, located in the bin directory of the Java distribution. The bin directory contains many additional tools for commandline processing.

- *TIP:* If, to your PATH environment variable, you add the path to `javac.exe` and `java.exe`, compiling and running Java programs from the console is much easier. Typical path to these executables is the following (modify the jdk version number as necessary)

```
C:\Program Files\Java\jdk-9.0.4\bin
```

  Add this string to the end of the PATH string in your system environment variables.

To **change** the system environment variables, follow these steps: Right-click Computer and click Properties. In the System Properties **window**, click Advanced System Settings > Environment Variables. Precede the path you are adding with a ';'

Compiling and Running from a Command Window

- Create a “hello world” example by typing the following lines of code in WordPad (or Notepad), and save in a convenient location as Hello.java.

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

- Open a command window, change directory to the directory that contains Hello.java, and then compile by typing the following command in the console:

```
javac Hello.java
```

- If a compiler error occurs, the compiler attempts to indicate clearly where the mistake occurred.

- When compilation is successful, a new file `Hello.class` has been created. Execute the code with the command

```
java Hello

//output:  Hello World!
```

Main Point

Java is an object-oriented programming language that is easier to use, less error-prone, more portable, and more popular than C++. Java code is compiled to *bytecode*, which can then be converted to native code on a target platform by way of a JVM interpreter. The inefficiency of an interpreter has largely been eliminated through the use of the *just-in-time compiler*, which compiles frequently occurring bytecode sequences to native code and caches these for further use, at runtime. Any language has the power to reveal or obscure the truth – as Maharishi says in SCI, "it is the power of speech that it can bind the boundless."

Integrated Development Environments

- A good IDE supports compiling, running, and debugging code with tools that are integrated and typically easy to use. For a large Java project, an IDE is indispensable.
- Good choices of IDE are NetBeans, IBM Rational Application Developer (formerly WebSphere Application Developer), Borland's JBuilder, JetBrains' IntelliJ
- Another excellent choice, which has become an industry standard, is the open-source IDE Eclipse, written entirely in Java. Among IDEs for Java, in recent years, Eclipse is the most widely used. We will use Eclipse in this course.

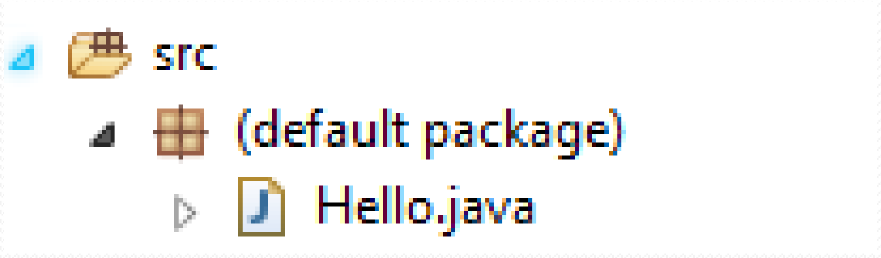
The Eclipse IDE

- Getting started. To use Java 9, you need Eclipse Oxygen (or later); earlier versions of Eclipse do not support Java 9.

- Features of the IDE [demo]

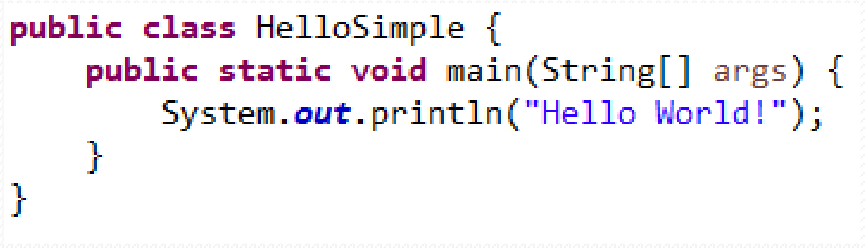
"Hello World" in Eclipse

- When you create a class Hello in Eclipse, it creates a default package. Later we will see how to place a class in a different package.

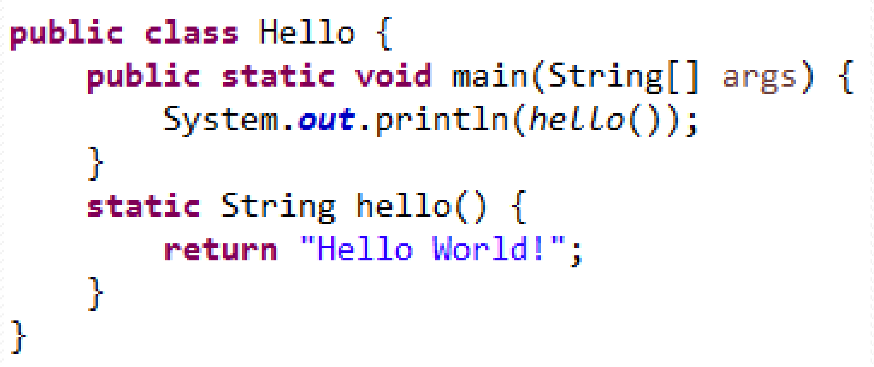


src
(default package)
Hello.java

- We can directly print "Hello World", or we can ask a function to return the String "Hello World" and then print that returned String



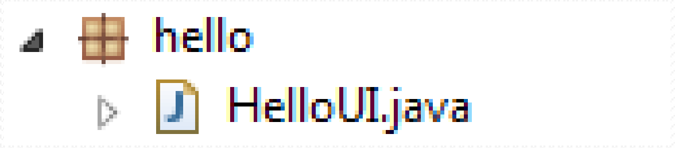
```
public class HelloSimple {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```



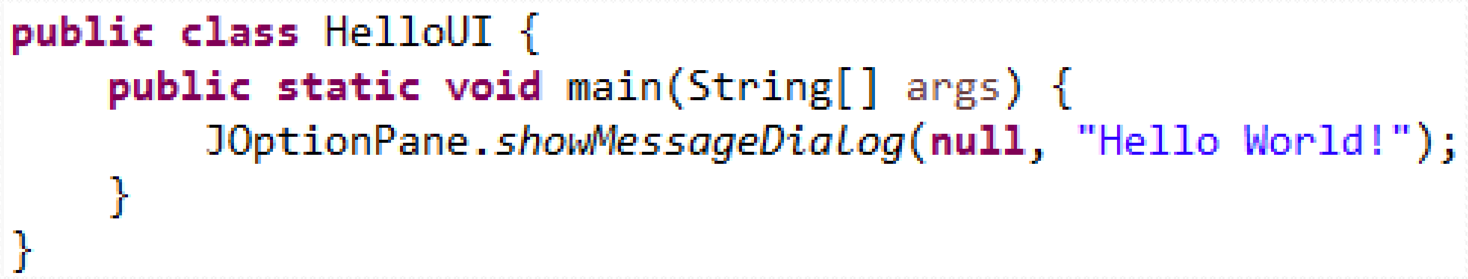
```
public class Hello {
    public static void main(String[] args) {
        System.out.println(hello());
    }
    static String hello() {
        return "Hello World!";
    }
}
```

"Hello World" continued

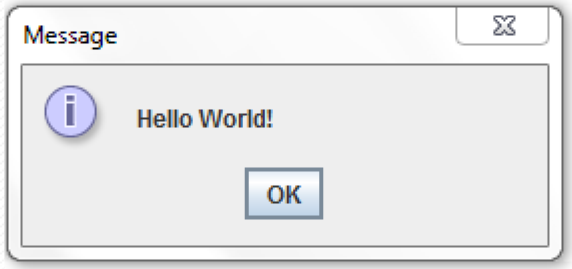
- This time we first create a package to contain our Hello class. We print "Hello World" this time using a Swing dialog box (Swing will be discussed in Lesson 5)



```
hello
    HelloUI.java
```



```java
public class HelloUI {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "Hello World!");
    }
}
```



Viewing Source Code

Allows you to see how Java has implemented their library code.

Steps:

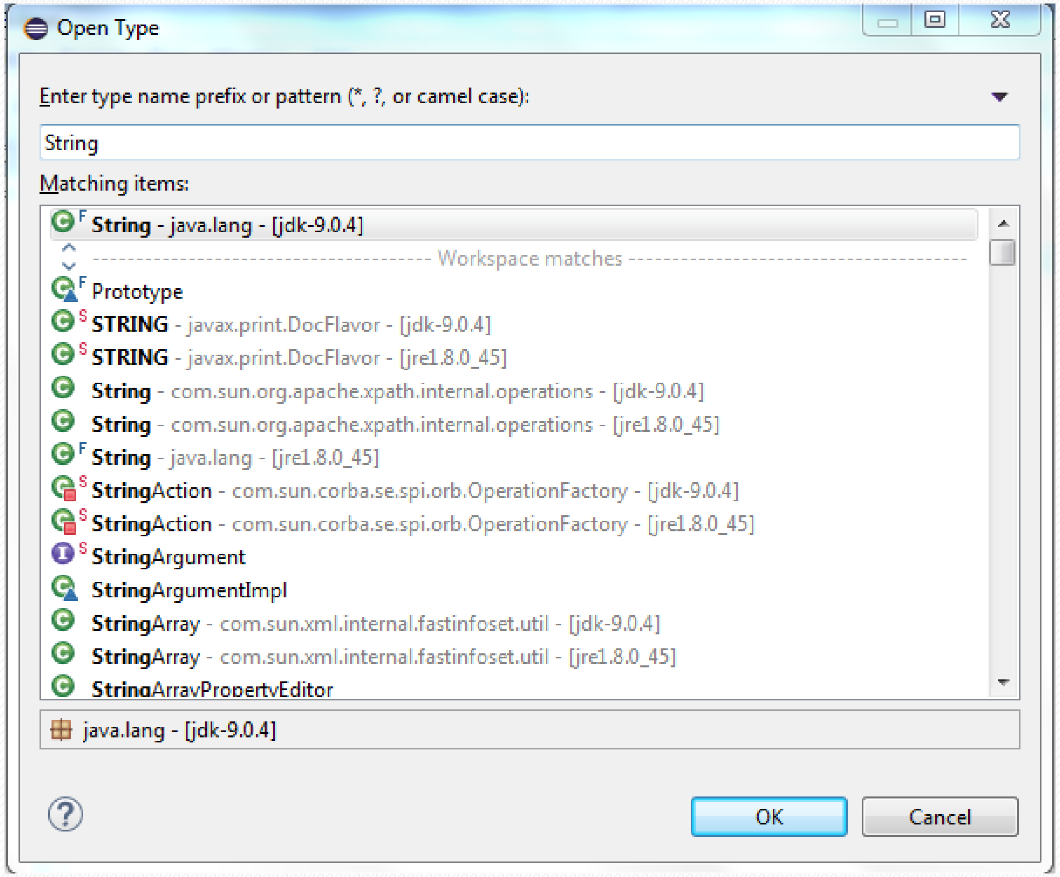
1. Open any project in Eclipse and type

   Ctrl Shift T

   Brings up this screen:

[Note: "String" was typed into search window]

(Caution: If you have multiple jre's installed, be sure to choose the Java 9 version of the desired class – in this case, String)



Open Type

Enter type name prefix or pattern (*, ?, or camel case):

String

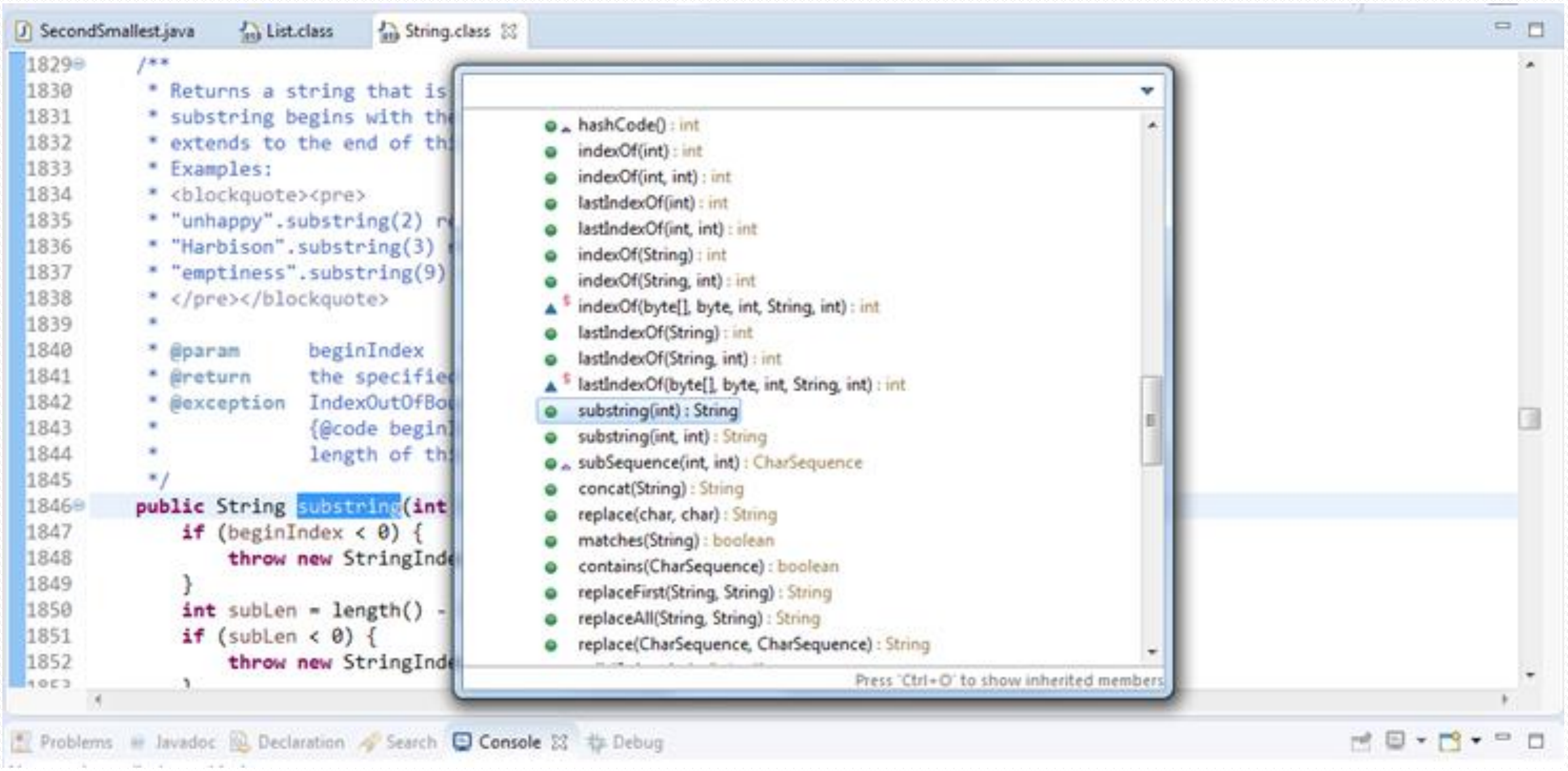
Matching items:

```
String - java.lang - [jdk-9.0.4]
------------------------------------ Workspace matches ------------------------------------
Prototype
STRING - javax.print.DocFlavor - [jdk-9.0.4]
STRING - javax.print.DocFlavor - [jre1.8.0_45]
String - com.sun.org.apache.xpath.internal.operations - [jdk-9.0.4]
String - com.sun.org.apache.xpath.internal.operations - [jre1.8.0_45]
String - java.lang - [jre1.8.0_45]
StringAction - com.sun.corba.se.spi.orb.OperationFactory - [jdk-9.0.4]
StringAction - com.sun.corba.se.spi.orb.OperationFactory - [jre1.8.0_45]
StringArgument
StringArgumentImpl
StringArray - com.sun.xml.internal.fastinfoset.util - [jdk-9.0.4]
StringArray - com.sun.xml.internal.fastinfoset.util - [jre1.8.0_45]
StringArrayPropertyEditor
```

java.lang - [jdk-9.0.4]

Viewing Source Code 2

- To locate a method inside the class (say, “substring()” in String), type Ctrl-O and type in search string



```
    /**
     * Returns a string that is
     * substring begins with the
     * extends to the end of thi
     * Examples:
     * <blockquote><pre>
     * "unhappy".substring(2) r
     * "Harbison".substring(3)
     * "emptiness".substring(9)
     * </pre></blockquote>
     *
     * @param      beginIndex
     * @return     the specifie
     * @exception  IndexOutOfBo
     *             {@code begin
     *             length of th
     */
    public String substring(int
        if (beginIndex < 0) {
            throw new StringInde
        }
        int subLen = length() -
        if (subLen < 0) {
            throw new StringInde
```

```
hashCode() : int
indexOf(int) : int
indexOf(int, int) : int
lastIndexOf(int) : int
lastIndexOf(int, int) : int
indexOf(String) : int
indexOf(String, int) : int
indexOf(byte[], byte, int, String, int) : int
lastIndexOf(String) : int
lastIndexOf(String, int) : int
lastIndexOf(byte[], byte, int, String, int) : int
substring(int) : String
substring(int, int) : String
subSequence(int, int) : CharSequence
concat(String) : String
replace(char, char) : String
matches(String) : boolean
contains(CharSequence) : boolean
replaceFirst(String, String) : String
replaceAll(String, String) : String
replace(CharSequence, CharSequence) : String
```

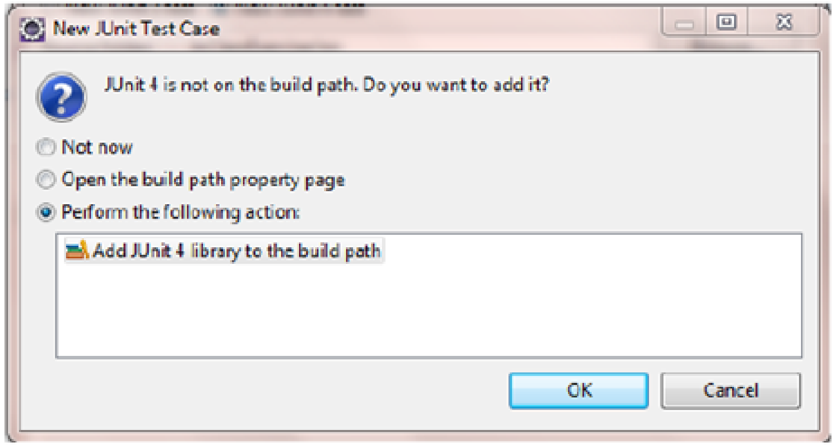
Press 'Ctrl+O' to show inherited members

Including Unit Testing in Your Work Environment

- A quick way to test your code as you develop is to run it from the main method as we have been doing
- A better way that is more reusable and useful for larger-scale team development is to have a parallel test project and use JUnit

Steps to set up JUnit

- Right click the default package where your Hello.java class is, and create a JUnit Test Case TestHello.java by clicking New→JUnit Test Case. (After we introduce the package concept, we will put the tests in a separate package.)
- Name it TestHello and click finish. You will see the following popup window.



New JUnit Test Case

JUnit 4 is not on the build path. Do you want to add it?

( ) Not now
( ) Open the build path property page
[✓ Perform the following action:]

Add JUnit 4 library to the build path

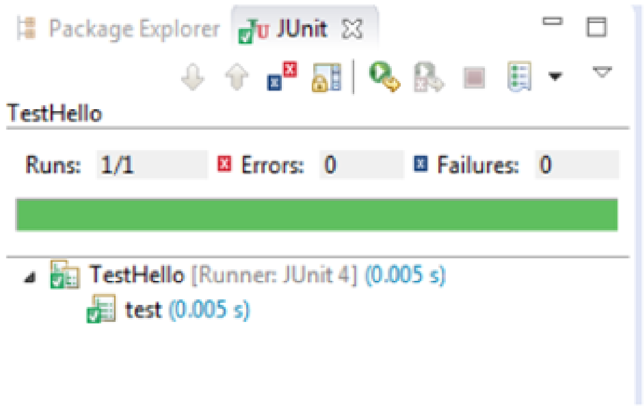
- Select "Perform the following action" → Add JUnit 4 library to the build path → OK
- Then you will see that JUnit 4 has been added to the Java Build Path by right clicking your project name → properties → Java Build Path.

Including Unit Testing in Your Work Environment

- Create a testHello method in TestHello.java (see below) and remove the method test() that has been provided by default:

```
@Test
public void testHello() {
    assertEquals("Hello", Hello.hello());
}
```

  If you have compiler error, you can hover over to the workspace where you have error and Eclipse will give you hints of what to do. (In this case, Add import 'org.junit.Assert.assertEquals'.)
- Run your JUnit Test Case by right clicking the empty space of the file → Run As → JUnit Test. If you see the result like below, it means you have successfully created your first unit test. ☺



```
TestHello
Runs: 1/1    Errors: 0    Failures: 0

TestHello [Runner: JUnit 4] (0.005 s)
    test (0.005 s)
```

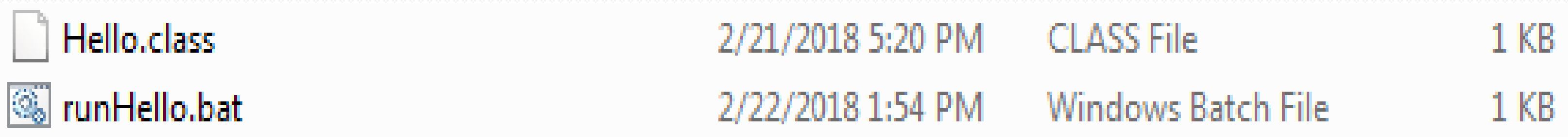
Deploying a Java Application

There are several ways to deploy a Java application. For this course, we demonstrate two of the ways:

- *Create a batch file* (Windows) to invoke the `main` method of the primary class of the application. (In Linux and Apple systems, a shell script can be used.)
- Save the application as a *runnable jar file*. The application can then be launched by double-clicking the jar file. (This approach is not good for applications that need to send output to the console.)

Creating a Batch File Launcher

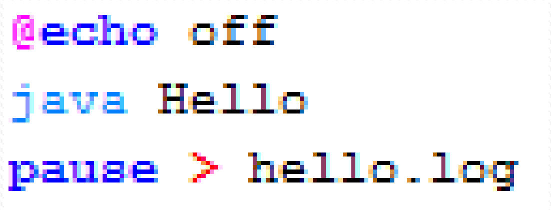
- If you used the default package, create a batch file in the same directory as the main `.class` file, like `Hello.class`. (Or, your batch file can cd to that directory as a first step.)



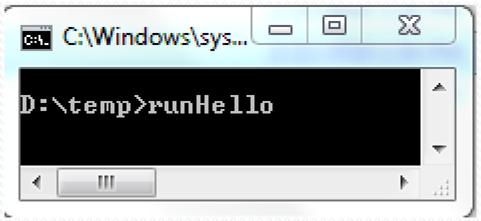
```
Hello.class      2/21/2018 5:20 PM   CLASS File           1 KB
runHello.bat     2/22/2018 1:54 PM   Windows Batch File   1 KB
```

- The batch file `runHello.bat` will consist of the commands used to launch the program from the commandline (as in earlier slides). You can launch the batch file by running it from the command window or by double-clicking the .bat file.

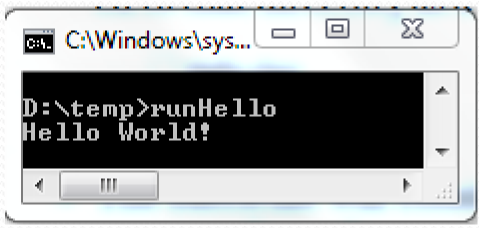
`runHello.bat`



```
@echo off
java Hello
pause > hello.log
```



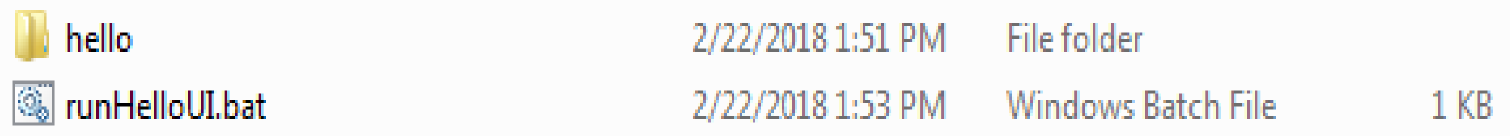
```
D:\temp>runHello
```



```
D:\temp>runHello
Hello World!
```

(continued)

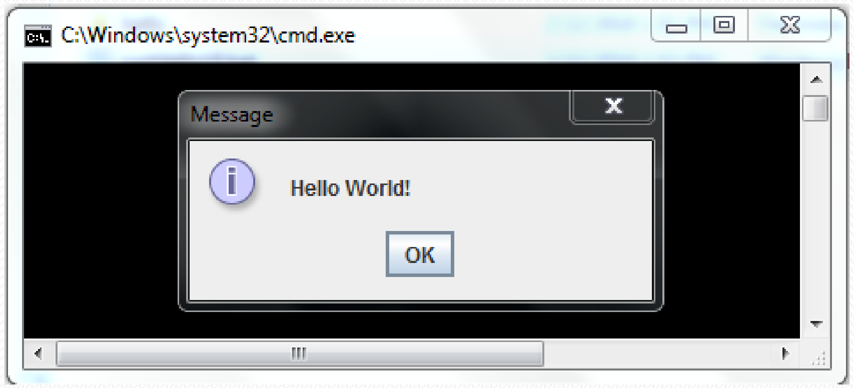
- If the `main` method is in a class that resides in a non-default package (as was done for `HelloUI` in package `hello`), place the batch file in the same directory as the top-level package.



hello	2/22/2018 1:51 PM	File folder	
runHelloUI.bat	2/22/2018 1:53 PM	Windows Batch File	1 KB

<u>`runHelloUI.bat`</u>

```
@echo off
java hello.HelloUI
```



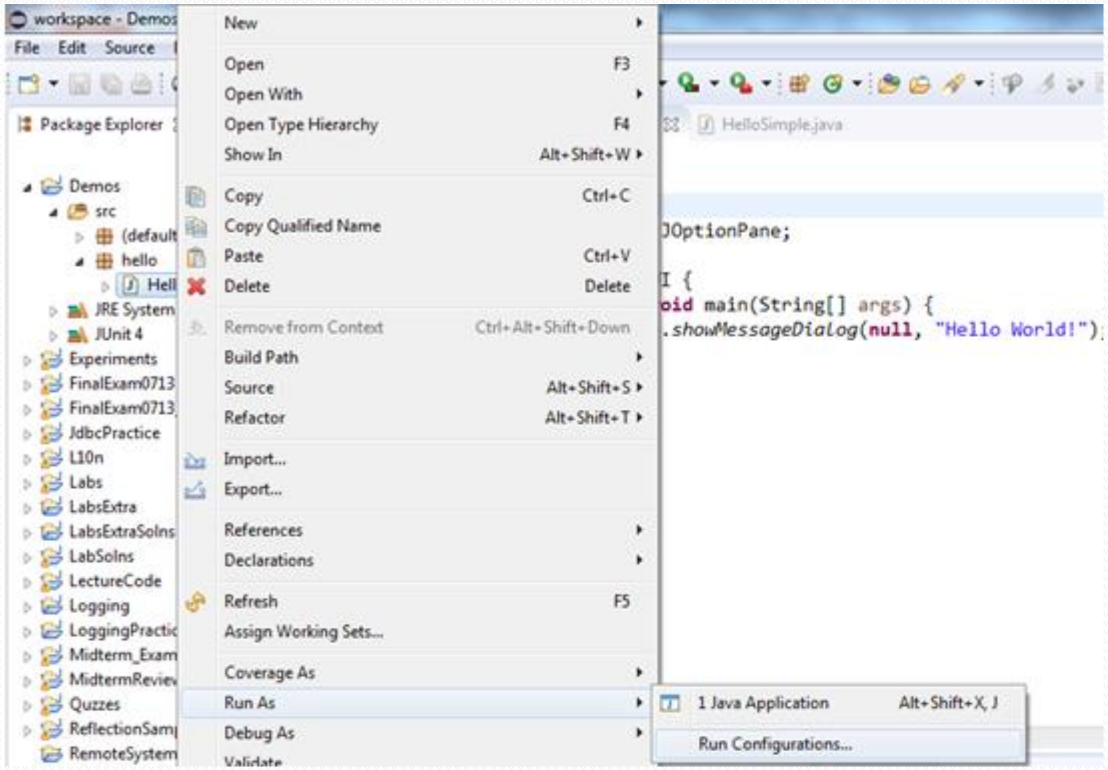
C:\Windows\system32\cmd.exe

Message

Hello World!

Creating a Runnable Jar

- *Create a Run Config.* Right-click on the class with the main method and select: Run As > Run Configurations …

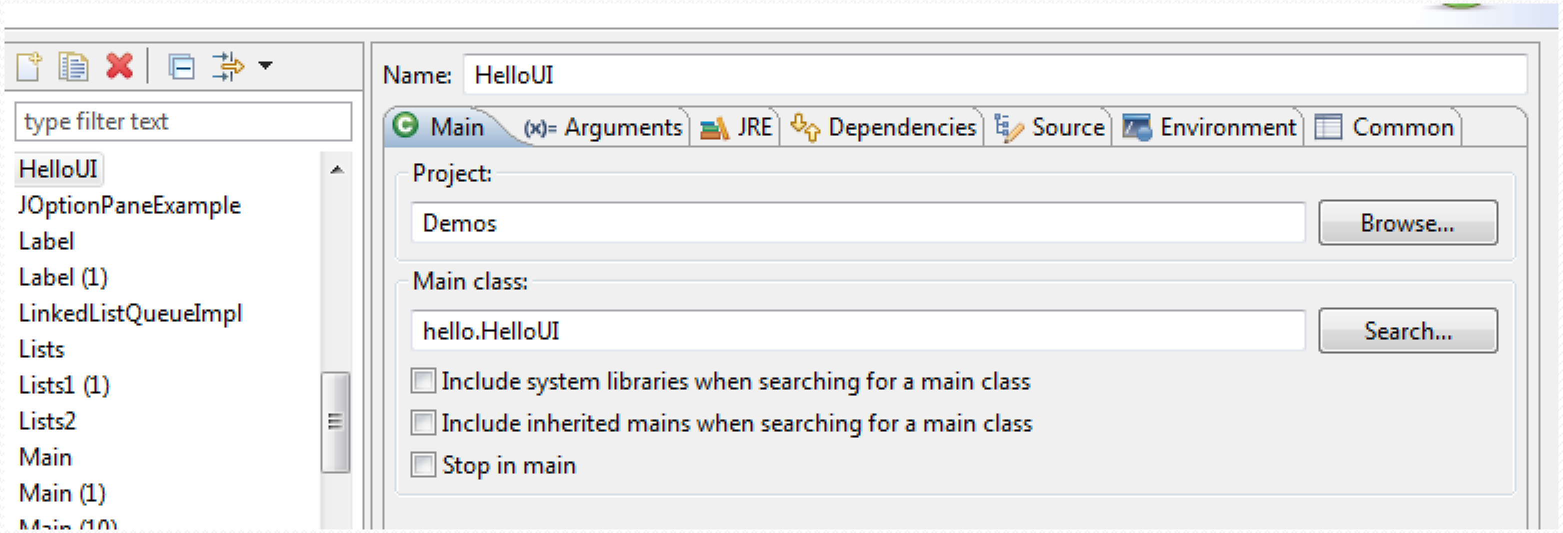


```
JOptionPane;

I {
oid main(String[] args) {
.showMessageDialog(null, "Hello World!");
```

Creating a Runnable Jar (2)

- Give the Run Config a name (here, it's HelloUI) and click the Close button



Name: HelloUI

Main | Arguments | JRE | Dependencies | Source | Environment | Common

Project: Demos

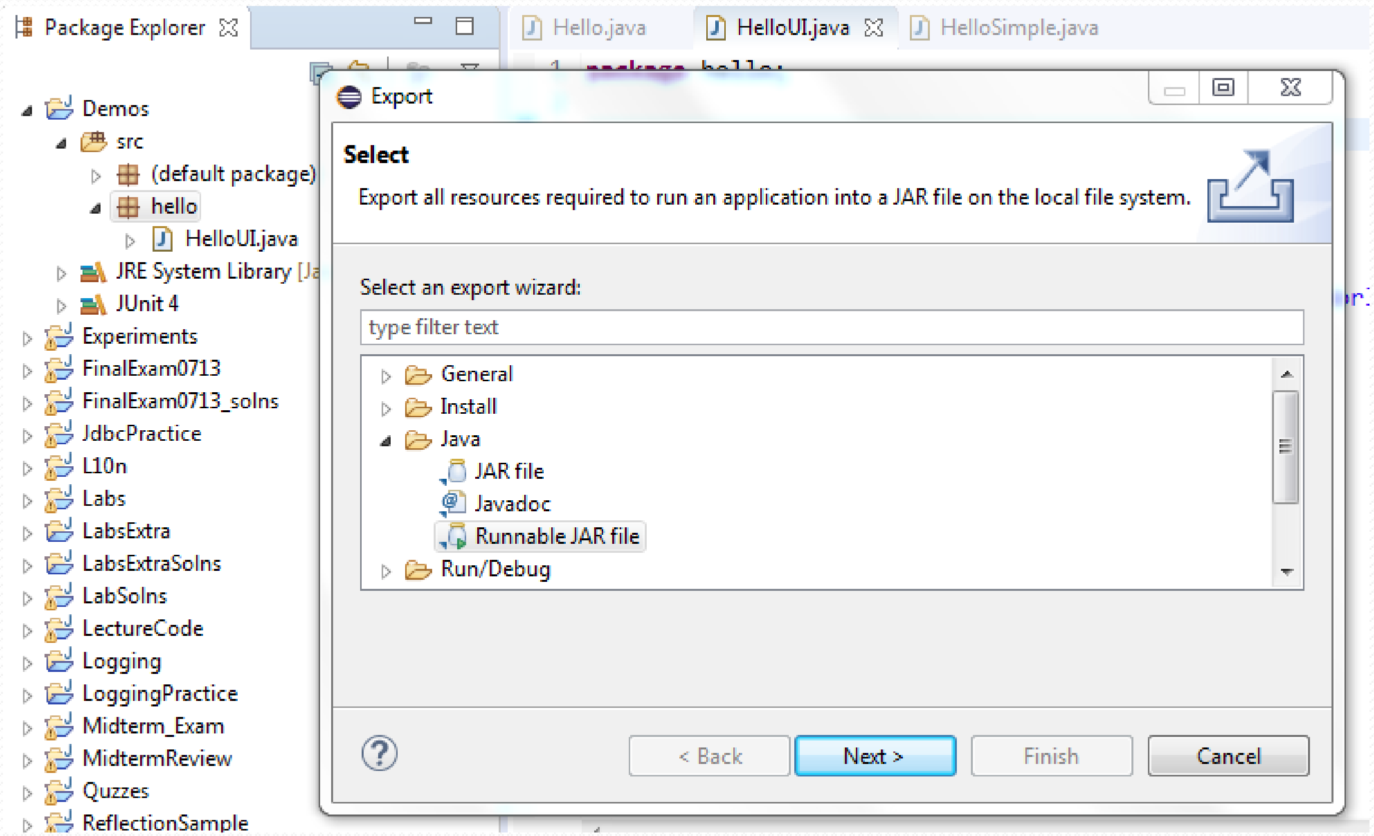
Main class: hello.HelloUI

- [ ] Include system libraries when searching for a main class
- [ ] Include inherited mains when searching for a main class
- [ ] Stop in main

Creating a Runnable Jar (3)

- Right-click on the top-level package and select

  Export > Java > Runnable JAR file



Select

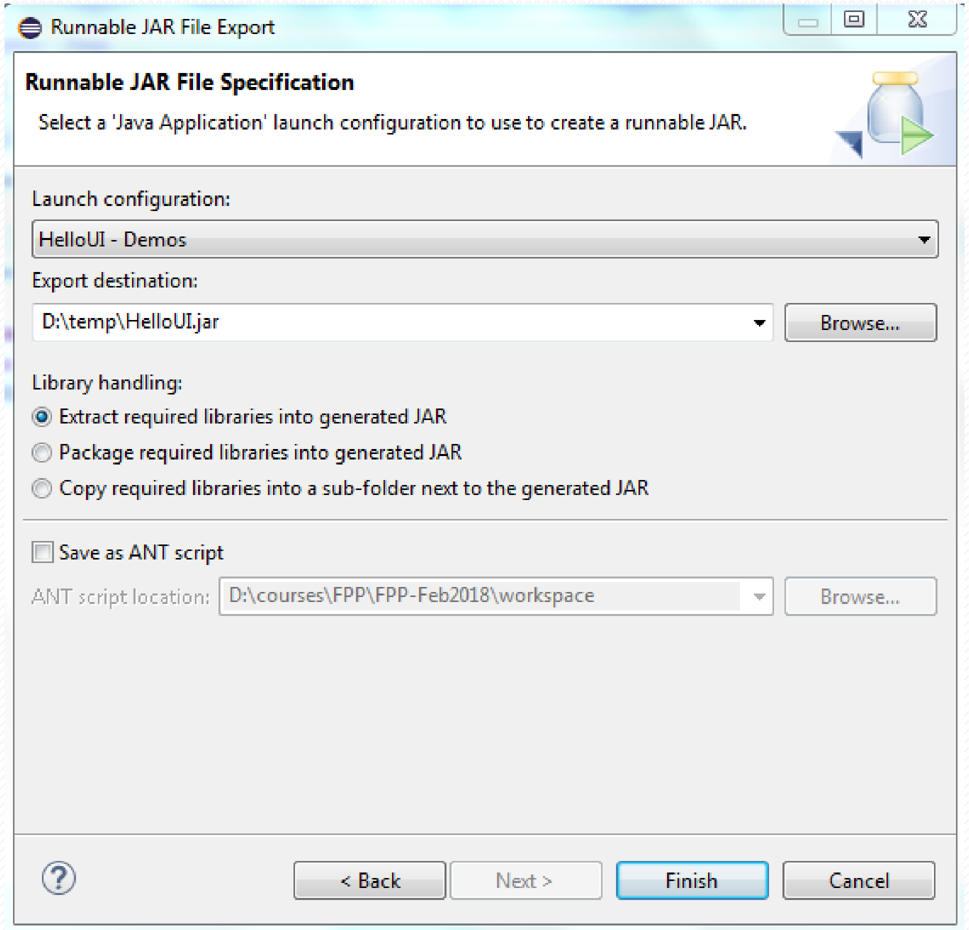
Export all resources required to run an application into a JAR file on the local file system.

Select an export wizard:

type filter text

Creating a Runnable Jar (4)

- Specify your Run Config and the destination directory



Runnable JAR File Export

Runnable JAR File Specification

Select a 'Java Application' launch configuration to use to create a runnable JAR.

Launch configuration: HelloUI - Demos

Export destination: D:\temp\HelloUI.jar

Library handling:

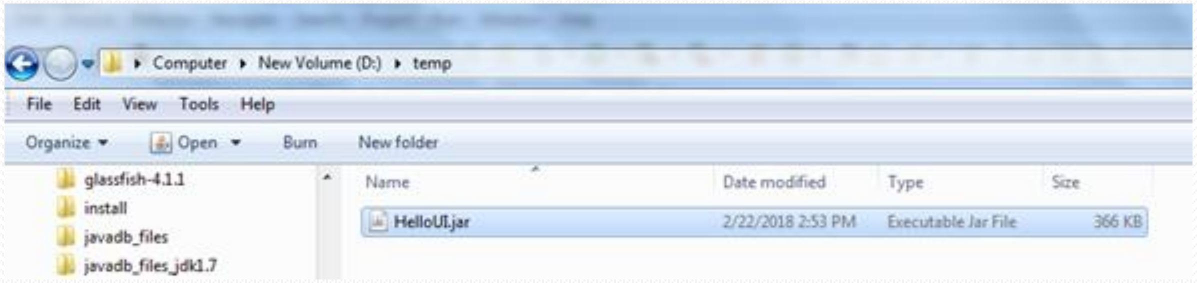
- [✓ Extract required libraries into generated JAR]
- Package required libraries into generated JAR
- Copy required libraries into a sub-folder next to the generated JAR

☐ Save as ANT script

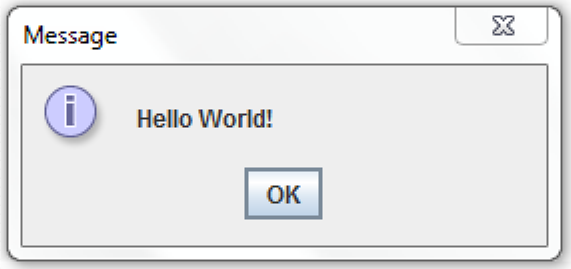
ANT script location: D:\courses\FPP\FPP-Feb2018\workspace

Creating a Runnable Jar (5)

- Now you can run your Java application by double-clicking on the jar file that your just created


Computer ▸ New Volume (D:) ▸ temp
File Edit View Tools Help
Organize ▾ Open ▾ Burn New folder
glassfish-4.1.1
install
javadb_files
javadb_files_jdk1.7

Name	Date modified	Type	Size
HelloUI.jar	2/22/2018 2:53 PM	Executable Jar File	366 KB



Hello World!

Main Point

Eclipse is a leading, open-source, 100% Java, integrated development environment, which provides excellent support for editing, compiling, running, and de-bugging Java applications. By analogy, to create a good life, we need to handle inner life and at the same time, structure a life-supporting environment – the goal is to live 200% of life.

Summary

- After the JDK has been installed, it is possible to use javac.exe and java.exe, located in the bin directory of the JDK distribution, to compile and run Java code.
- Developing code is much easier using an IDE – the most widely used IDE for Java is Eclipse.
- Eclipse allows you to edit, compile, debug, and run Java code in an integrated environment. It also lets you view Java library code (ctrl-shift-T, and attach source).
- JUnit is one of the most popular Java add-ons for unit-testing Java code, and is supported in the Eclipse environment.
- Java applications may be deployed either by creating a batch file (Windows only) that calls the main method of the main class, or creating a runnable jar file using tools provided by Eclipse.

Connecting the Parts of Knowledge With the Wholeness of Knowledge

1. Using Java, highly functional applications can be built more quickly and with fewer mistakes than is typically possible using C or C++.
2. To optimize the use of Java's features, IDE's such as Eclipse ease the work of the developer by handling in the background many routine tasks.

---

3. **Transcendental Consciousness:** To be successful, action must be based on the field of pure intelligence, which is located at the source of thought.

4. **Wholeness moving within itself**: In Unity Consciousness, the pure intelligence located in TC is found pervading all of creation, from gross to subtle.