

MAHARISHI UNIVERSITY of MANAGEMENT

Engaging the Managing Intelligence of Nature

Computer Science Department

**CS390 Fundamental Programming
Practices (FPP)**

**Professor Paul Corazza and
Professor Ankhtuya Ochirbat**

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

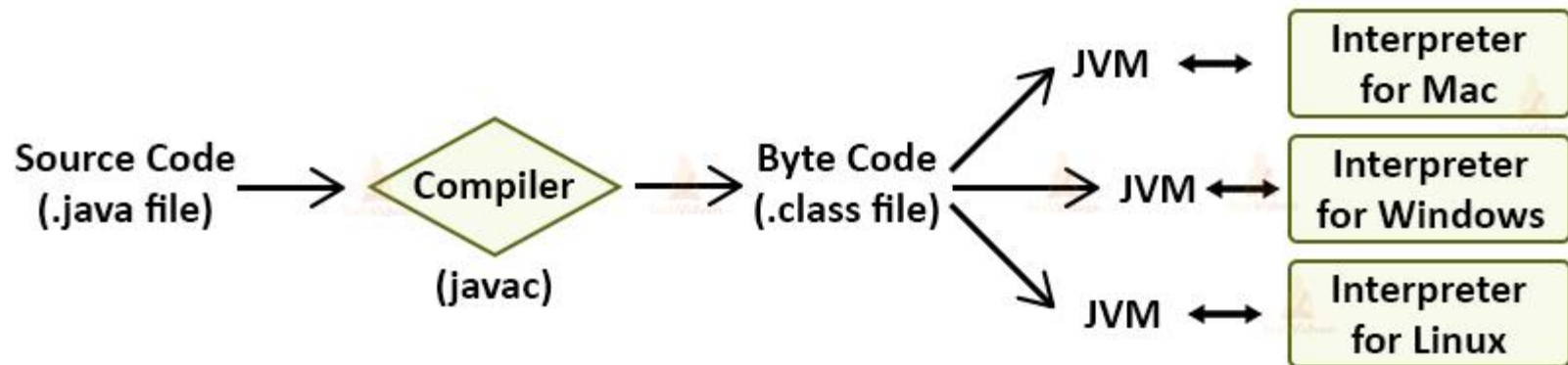
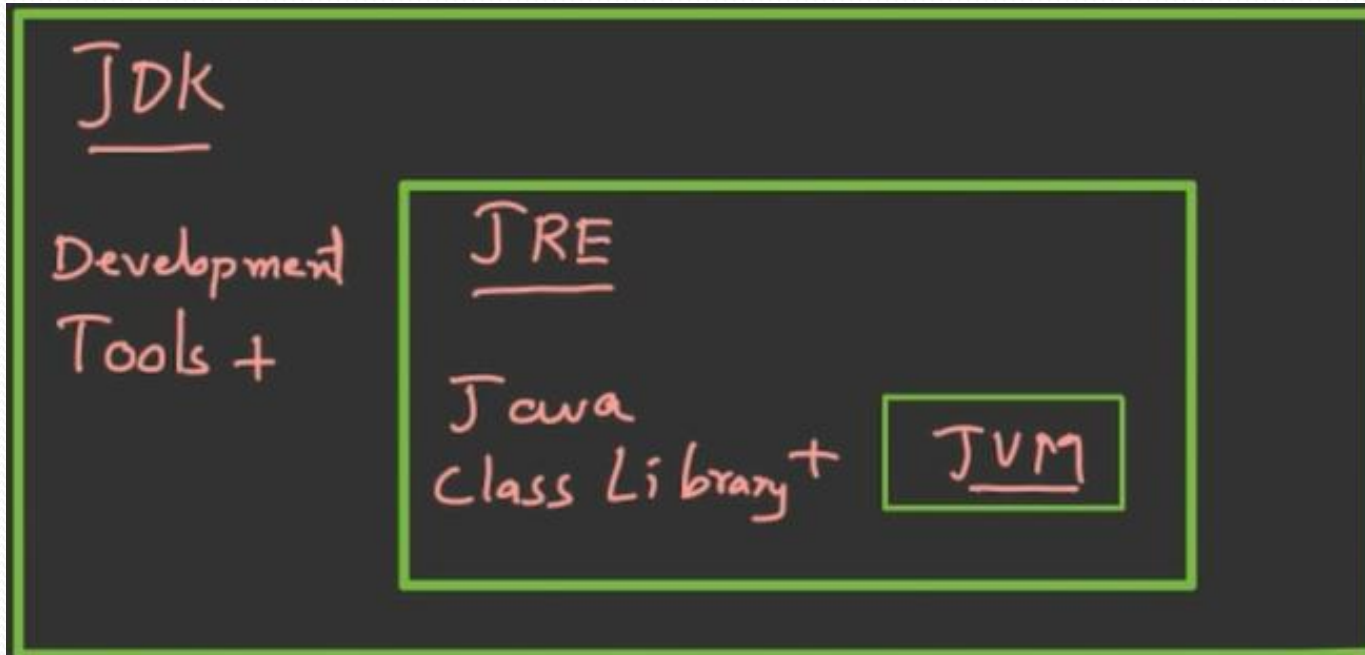
- ***Brief History.*** The Java language began as a language for programming electronic devices, though the original project was never completed. Its creator was **James Gosling**, of Sun Microsystems. The language was developed privately starting in 1991, and was made publicly available in 1994. In 2009, Oracle bought the rights to Java from Sun Microsystems.
- ***Java Is an OOP Language.*** Java is an object-oriented programming language. This means that Java programs are designed by defining software objects that interact with each other (mirroring the way things get done in the real world).
- ***Number 1 Language.*** For more than 20 years, Java has been the Number 1 programming language in the IT world.

The Java API Docs

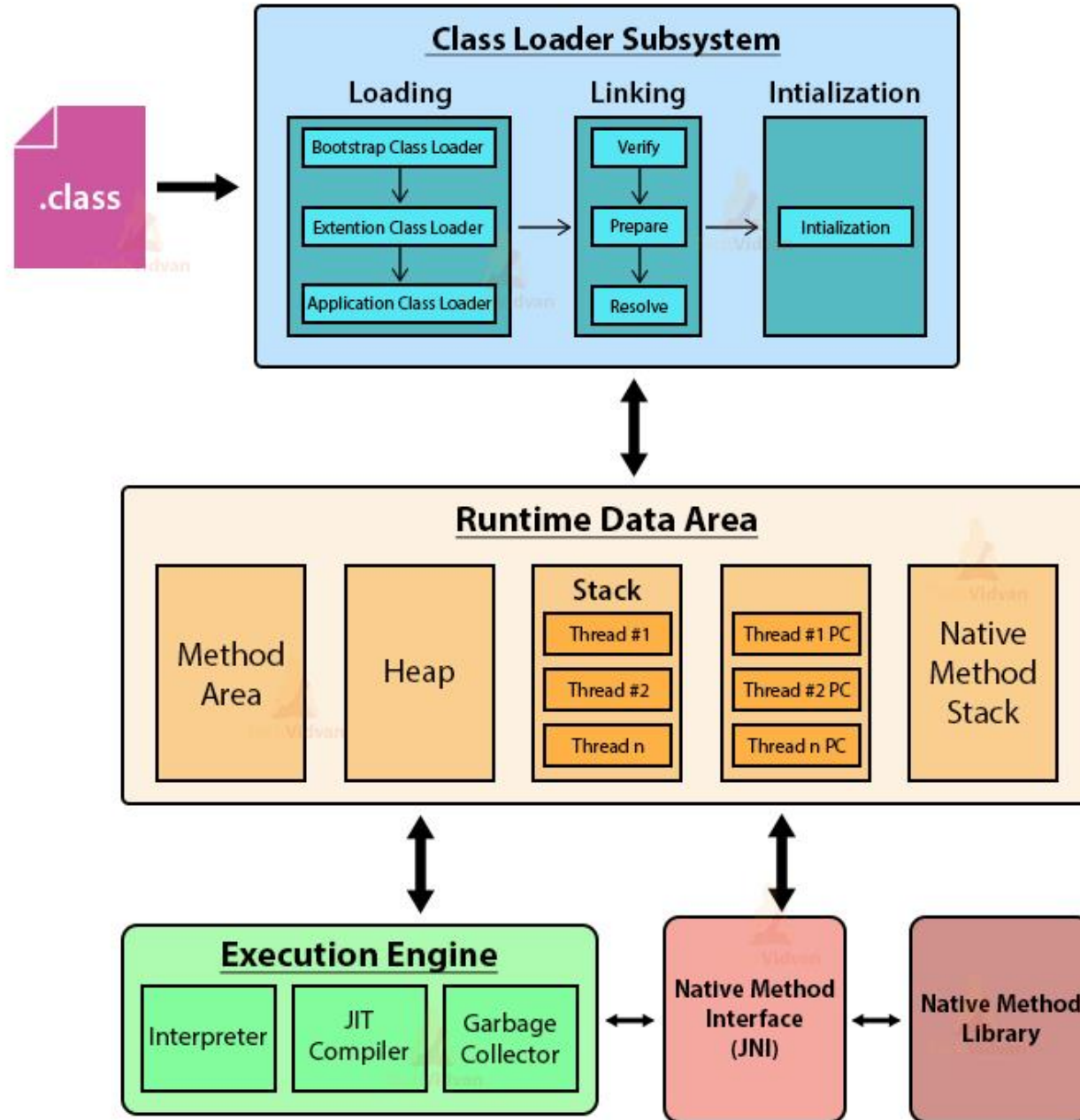
- Oracle provides online documentation of all the Java library classes. Full documentation of each class in the Java libraries is provided. For Java 17, the link is

<https://docs.oracle.com/en/java/javase/17/docs/api/index.html>

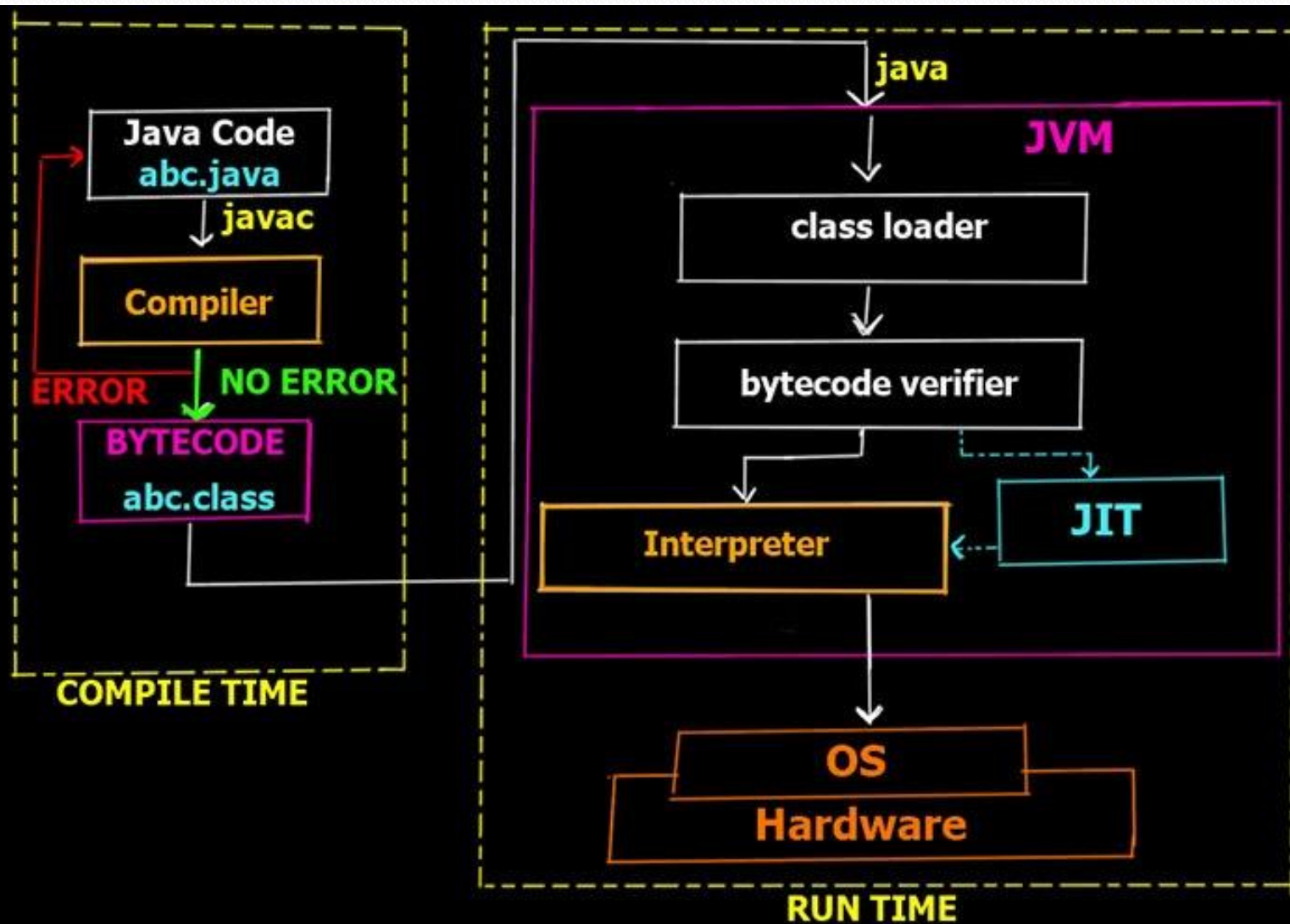
JDK, JRE, and JVM



JVM Architecture



[Ref]





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 been an industry standard for many years, 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.* For this course, you will need one of the recent versions of the JDK and of Eclipse.
- Features of the Eclipse IDE [demo]
 - Workspace > project > package > class
 - "Perspectives" in Eclipse: Java and Debug perspectives
 - Setting preferences
 - Save / compile / run / debug (example: HelloWorld)
 - Auto-formatting

"Hello World"

- Create the simplest Java program. This involves creating a package and then defining a class inside that package.

```
package lesson1.hello;
```



```
public class HelloWorld {
```

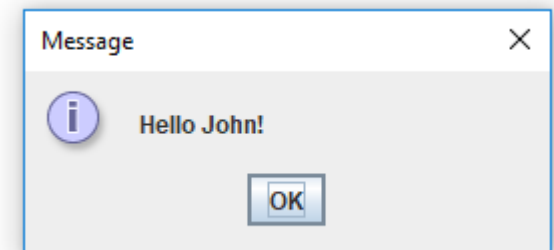
```
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }
```

```
}
```

"Hello World"

- We create a simple Java program. This involves creating a package and then defining a class inside that package.

 lesson1.hello
>  Hello.java



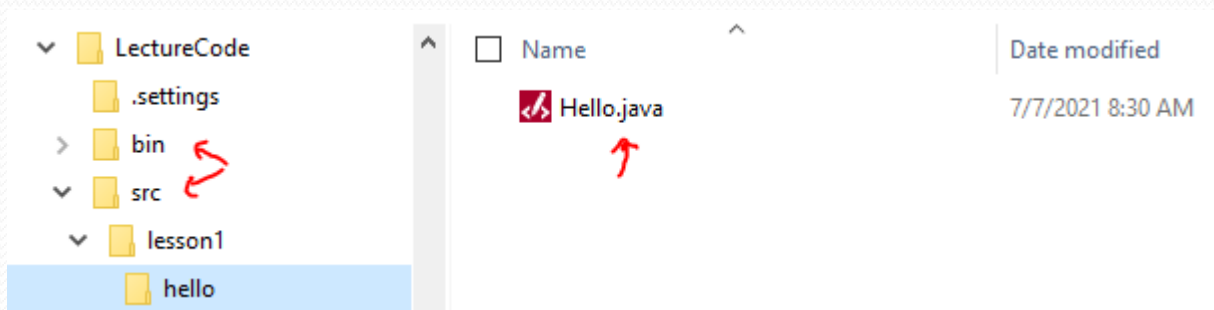
```
package lesson1.hello;

import javax.swing.JOptionPane;

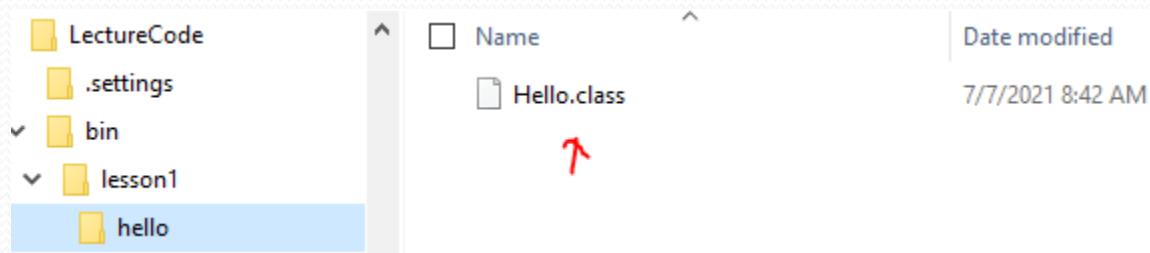
public class Hello {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "Hello John!");
    }
}
```


"Hello World" – the class file

When you create an Eclipse project, Eclipse creates a src folder (which will contain your source code) and a bin folder (which will contain the compiled files – the .class files).



The .class file is an intermediate file consisting of *bytecode*. The bytecode is interpreted by the operating system into native code at runtime.



In-Class Exercise 1.1

- In this exercise you will run a Java application `SampleClass` within your Eclipse workspace, available in the `InClassExercises` project.
- The comments written in the class `SampleClass` explain many details about basic Java code. The goal is to study the code and run it to get a feeling for how it works, and then try to use what you have learned to implement the requirements in a second Java class `MyClass`.

Testing Code Using JUnit

- JUnit is a third-party Java component that can be used to test your code.
- In the SampleClass file, we might want to test the method

```
int calculateSum(int[ ] arr)
```

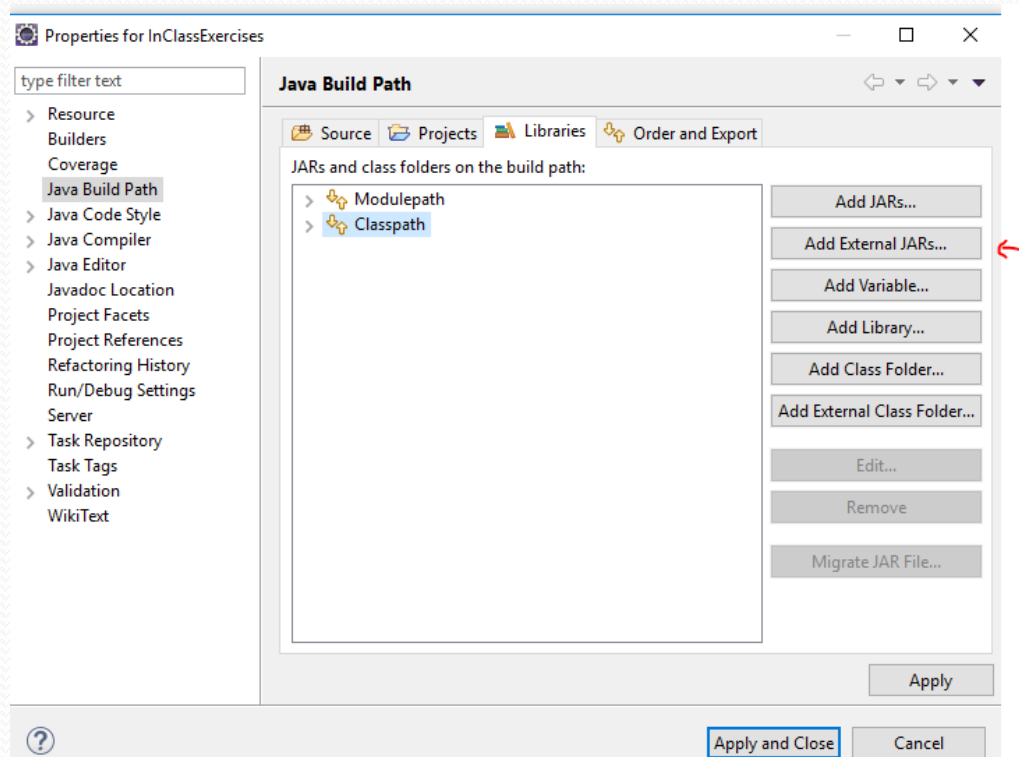
- To do this, we create a TestSampleClass class with a method

```
void testCalculateSum()
```

- Testing with JUnit requires us to load the JUnit libraries into our Eclipse project.

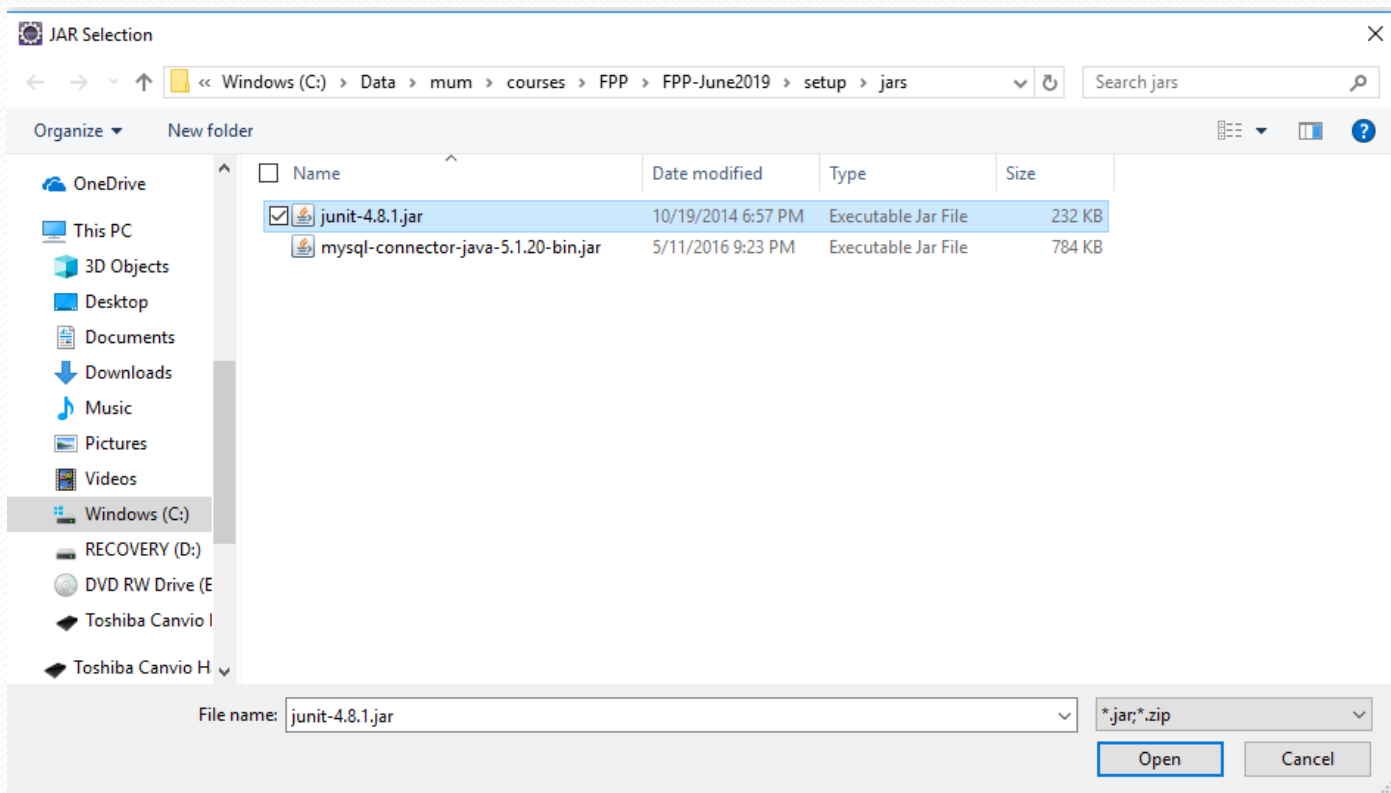
Adding JAR Files in Eclipse

In order to make use of the JUnit testing tool, you need to add junit-4.8.1.jar (or a more recent version) to the "classpath". Do the following: Right click on the Project (here, it is InClassExercises) and select Properties > Java Build Path, select Classpath, and click Add External Jars...



Adding JAR Files in Eclipse

- Navigate to junit-4.8.1.jar in fpp-setup (this is found in the Set-up zip file in Resources), select and click Open



Then click
Apply and
Close

Using JUnit – Testing SampleClass

```
import static org.junit.Assert.assertTrue;
import org.junit.Test;

public class TestSampleClass {
    @Test
    public void testCalculateSum() {
        int[] testArray = {1, 2, 3};
        int result = SampleClass.calculateSum(testArray);
        int expected = 1 + 2 + 3;
        assertTrue(expected == result);
    }
}
```

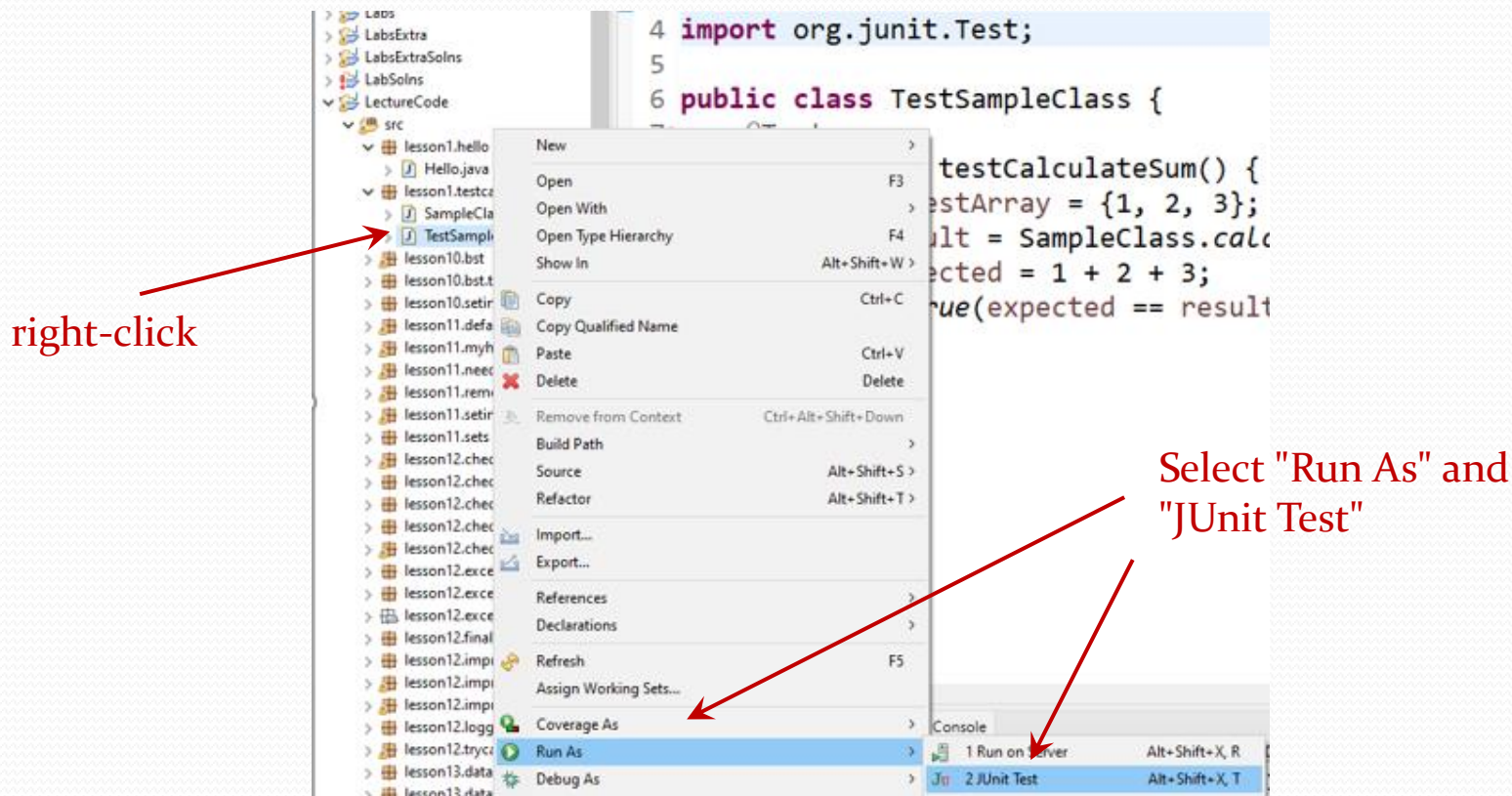
tells JUnit to run this test, uses import

- method from JUnit library
- accepts a boolean and expects that this boolean evaluates to *true*
- if boolean is false, JUnit declares that the test failed
- static import used to make accessible

- In a test class, you can ask JUnit to test a method of your class by marking the test method with *@Test*
- Within the test method, set up some test data, run the test class's method on this data and compare the result to the correct answer by using JUnit's *assertTrue* (or *assertEquals*) method
- If class's computation does not match expected result, test fails.
- See Demo

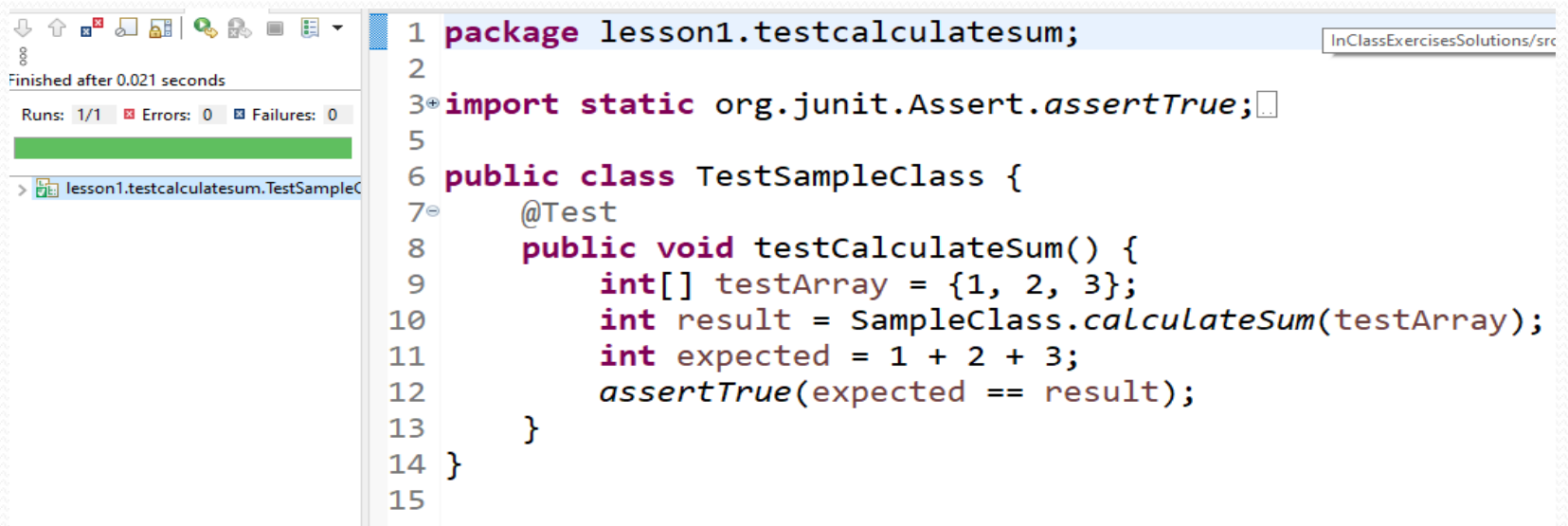
Using JUnit in Eclipse

- To run your test, right click on the file TestSampleClass.java and select Run As / JUnit Test.



Using JUnit in Eclipse

- If your code passes the test, you will see something like the following:



The screenshot shows the Eclipse IDE interface. On the left, the 'Run' console displays the message 'Finished after 0.021 seconds' and 'Runs: 1/1', 'Errors: 0', 'Failures: 0'. Below this, the 'Test Explorer' shows a tree view with 'lesson1.testcalculatesum.TestSampleC'. The main editor window displays the following Java code:

```
1 package lesson1.testcalculatesum;
2
3 import static org.junit.Assert.assertTrue;
4
5
6 public class TestSampleClass {
7     @Test
8     public void testCalculateSum() {
9         int[] testArray = {1, 2, 3};
10        int result = SampleClass.calculateSum(testArray);
11        int expected = 1 + 2 + 3;
12        assertTrue(expected == result);
13    }
14 }
15
```

In-Class Exercise 1.2

- In this exercise you will create a test class `TestMyClass` to test the two methods in `MyClass`, which you implemented in Exercise 1.1.
- Be sure to add the JUnit JAR file to the project before trying to access the `@Test` annotation and the `assertTrue/assertEquals` method.

Main Point

Eclipse is a leading, open-source, integrated development environment, which provides excellent support for editing, compiling, running, and debugging 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.

JShell - Isolating a Problem

To test a small piece of Java code, you may wish to compile and run it separately from your program or IDE. One of the ways that this can be done is by using Jshell, a commandline feature of Java. To use JShell, go to a command window, enter JShell, and hit return. In order for this to work properly, you *might* need to set up the JDK command line tools

Trying Out Code with JShell

- JShell is a Java 9 feature that lets you try out code interactively at the commandline.
- To start JShell, type `jshell` at the commandline

```
C:\Users\paul_>jshell
| Welcome to JShell -- Version 16.0.1
| For an introduction type: /help intro

jshell>
```

- You can use JShell to evaluate expressions

```
C:\Users\paul_>jshell
| Welcome to JShell -- Version 16.0.1
| For an introduction type: /help intro

jshell> 2 * 3 + 7
$1 ==> 13
```

(continued)

- You can define a Java method in JShell and then run it

```
jshell> String hello() {  
    ...>     return "Hello World!";  
    ...> }  
| created method hello()  
  
jshell> hello()  
$3 ==> "Hello World!"  
  
jshell>
```

- Exit JShell by typing /exit at the jshell prompt

```
jshell> /exit  
| Goodbye
```

Viewing Java Library Source Code

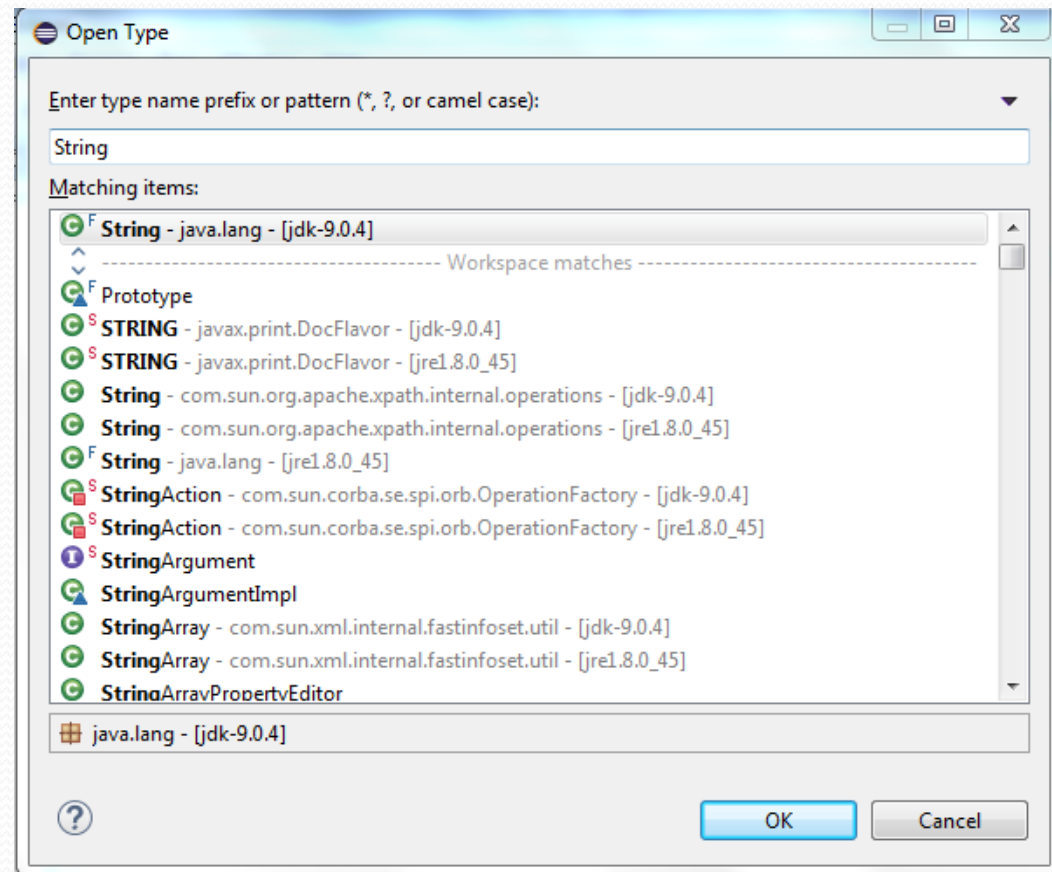
Allows you to see how Java has implemented their library code. You must have src.zip in your Java distribution. (The Java installer automatically provides src.zip)

Steps:

1. Open any project in Eclipse and type Ctrl Shift T
Brings up this screen:

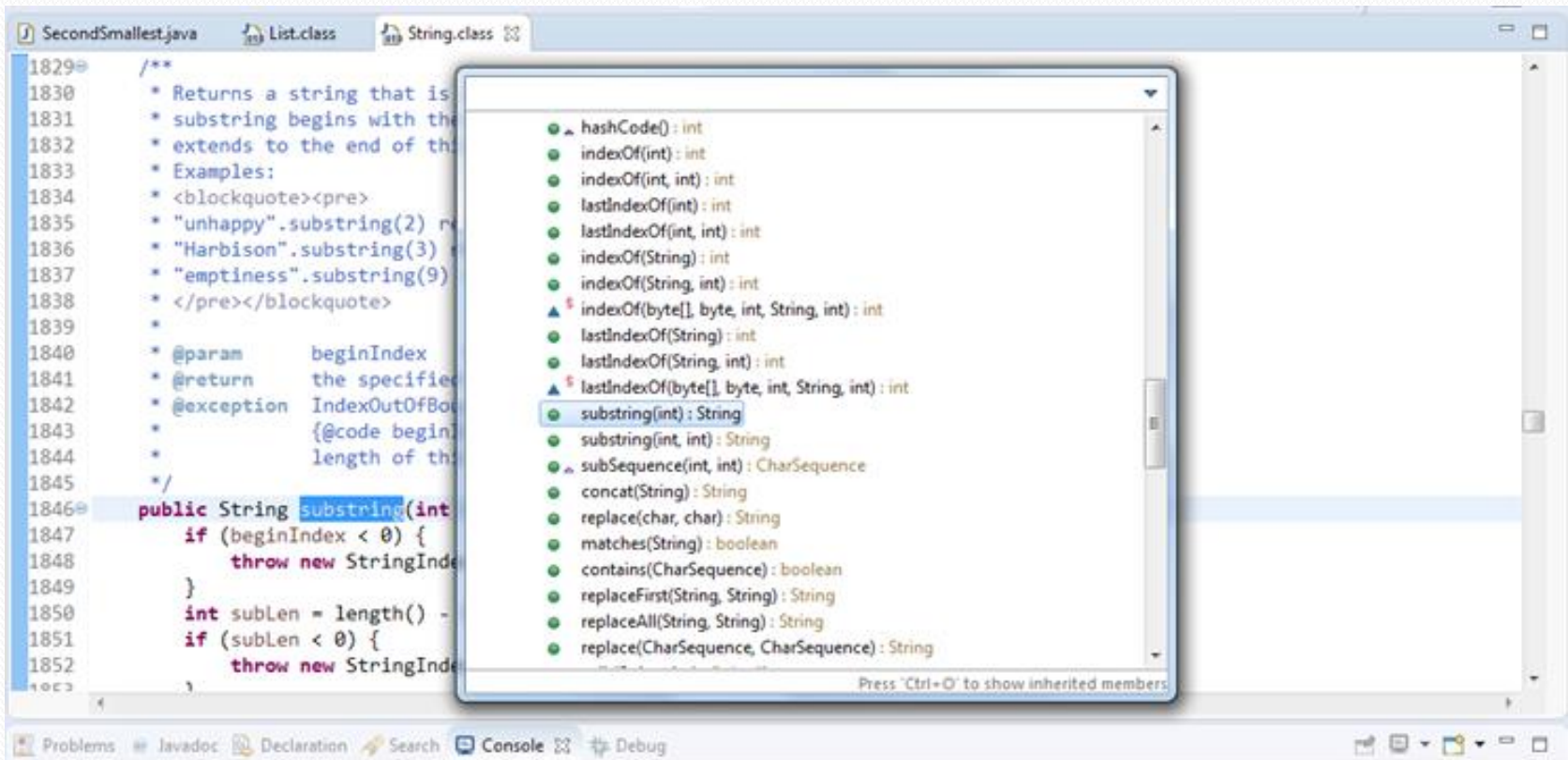
[Note: “String” was typed into search window]

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



(continued)

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



Deploying a Java Application

While code is being developed, you run your code within the IDE. But when your code needs to be delivered ("deployed"), you need to have another way of launching it. There are several ways to do this. For this course, we demonstrate one of the ways:

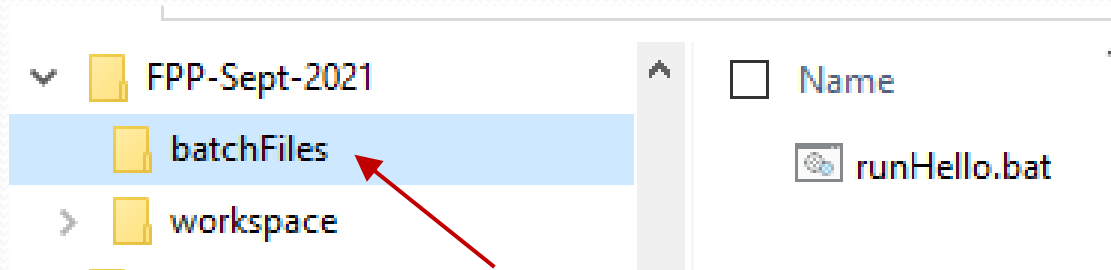
- *Create a batch file* (Windows) to invoke the `main` method of the primary class of the application. (In Linux and Apple systems, a shell script can be used.)

Steps: Your batch file needs to:

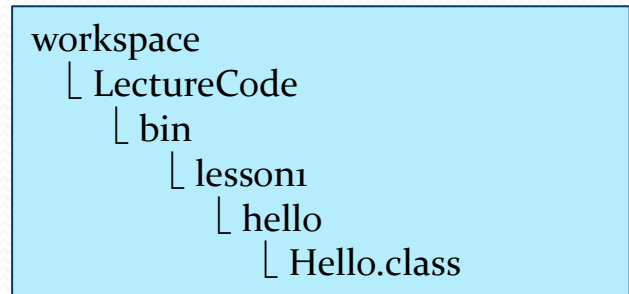
1. Navigate to the full name of the application
(for our class, the application will live in the bin dir)
2. Call the `java.exe` on the fully qualified java filename

Creating a Batch File Launcher for Hello

Our batch file example lives in *batchFiles*. The batch file needs to locate our application `lesson1.hello.Hello.class`. See demo



batch file located here



```
@echo off
cd ..\workspace\LectureCode\bin
java lesson1.hello.Hello
```

navigate to program location

call java.exe to execute program,
using fully qualified name

(continued)

- To execute the batch file, either double click the .bat file or call from the command prompt.

```
C:\Users\paul_>cd C:\Data\mum\courses\FPP\FPP-Sept-2021\batchFiles  
C:\Data\mum\courses\FPP\FPP-Sept-2021\batchFiles>runHello.bat
```



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, IDEs 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.