

# Introduction

Dr. Tamer ABUHMED  
Java Programming Course (SWE2023)  
College of Computing

# Course Overview



- **Staff**

- ABUHMED, Tamer - 타메르 (tamer@skku.edu)
- TA - TBD

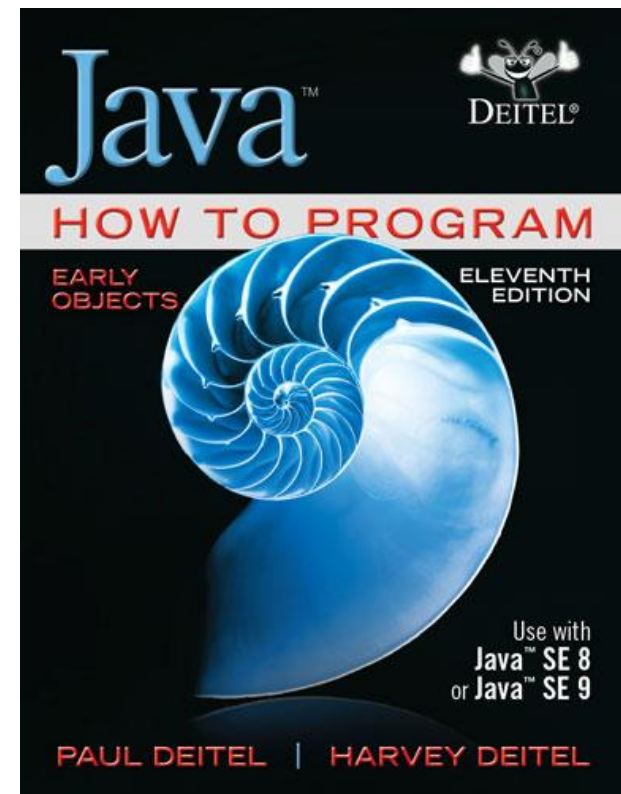
- **Textbook:** Java How to Program (ed. 10 or 11)

- **Grading Policy**

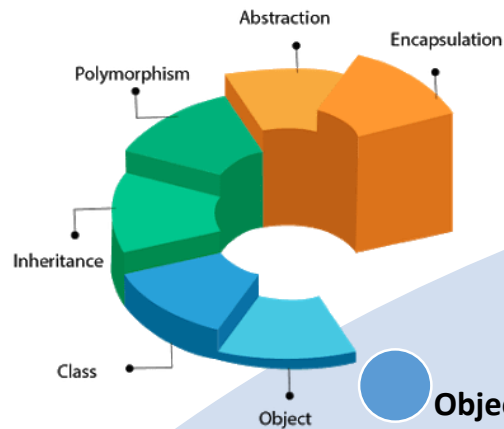
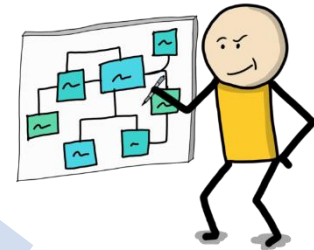
Assignments (40%), Final exam(30%),  
Attendance (10%), **Class Projects** (20%)

- **Cheating Policy**

- Automatic **F** for both



# Course Plan



## Object-oriented Programming

- Classes
- Inheritance
- Polymorphism
- Abstraction
- Exceptions

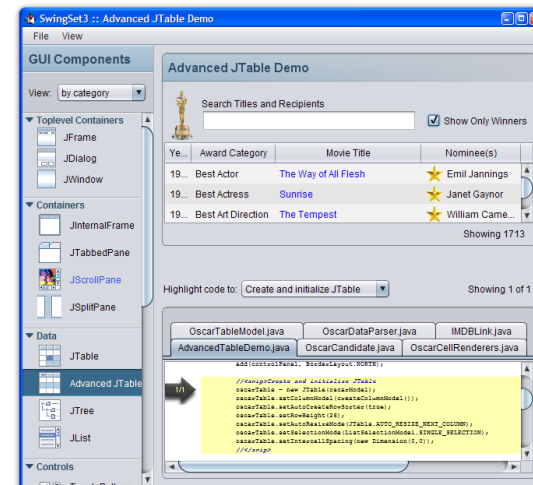
## Java basics for C++ programmers

## Introduction

## Graphical User Interface

## Advance Topics

- Generic Collections
- Files, Streams and Object Serialization
- Java Network Programming



# Important!



- This class is about “Java programming”
- Java syntax will be discussed, but you will have to learn the basics stuffs (if .. Else, loop, functions, etc.) by yourself.

## Prerequisite(very strict)

- Object oriented programming C++
- Comfortable in programming using at least one language

## Caution!

- This class will require intensive java programming for a skillful programmer (more than 5 hours per week)

# Introduction To Java



- Most people are familiar with Java as a language for Internet applications



Applet  
Servlet  
Java Server Pages (JSP)

- We will study Java as a general purpose programming language
  - The syntax of expressions and assignments will be similar to that of other high-level languages such as C++.
- Java is the world's most widely used computer programming language.

# Java Applications



- **Standalone applications**
  - Console Applications
  - Swing Applications (GUI)

- **Web applications**

- Applet
- Servlet
- JSP

- **Mobile applications**

- J2ME Applications
- Android Applications

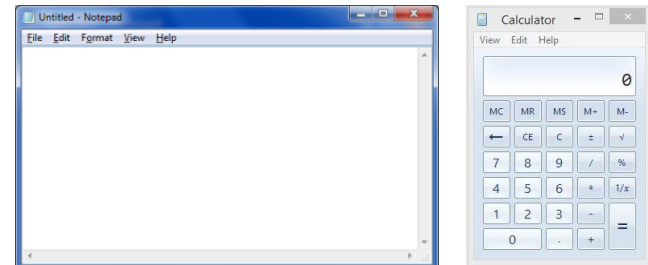


- **Distributed Applications**

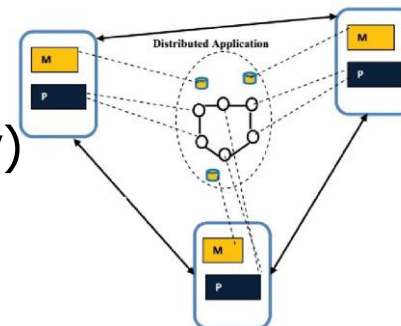
- Enterprise Java Beans (EJB Technology)

- **Embedded Systems Applications**
- **Java Card Applications**

- smart card programming



Swing Applications (GUI)



# Origins of the Java Language



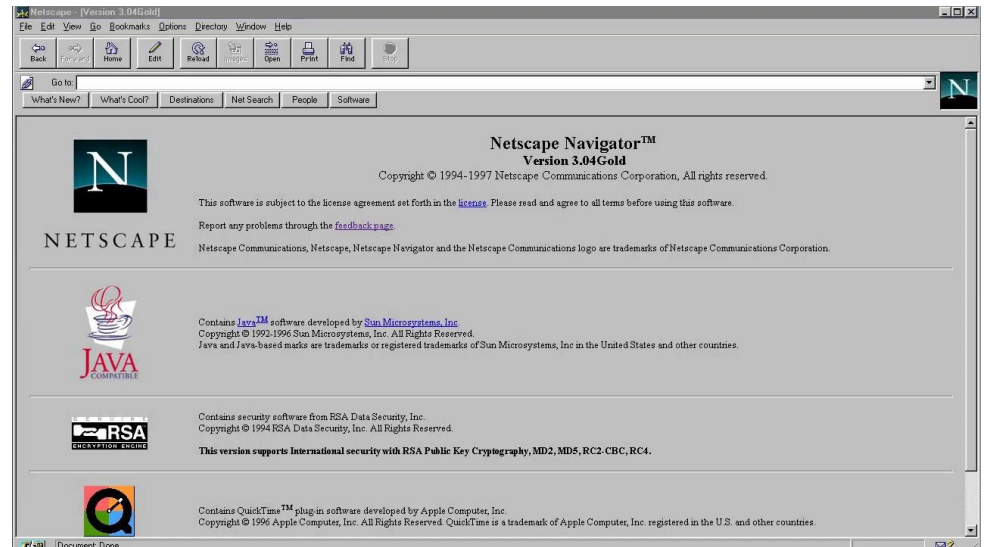
- Created by Sun Microsystems team led by James Gosling (1991)
- **Why is it called Java?**
- First name of this language was Oak
- The name was existed, then it was renamed as Java.
- **Originally designed** for programming home appliances
  - Difficult task because appliances are controlled by a wide variety of computer processors
  - Team developed a two-step translation process to simplify the task of compiler writing for each class of appliances



# Origins of the Java Language



- Patrick Naughton and Jonathan Payne at Sun Microsystems developed a Web browser that could run programs over the Internet (1994)
  - Beginning of Java's connection to the Internet
  - Original browser evolves into HotJava
- Netscape made its Web browser capable of running Java programs (1995)
  - Other companies follow suit





# Java is everywhere!



- Sun Microsystems was acquired by Oracle in 2009.
- Based on [www.java.com](http://www.java.com) website statistics 2016:
  - ✓ 97% of Enterprise Desktops Run Java
  - ✓ 89% of Desktops (or Computers) in the U.S. Run Java
  - ✓ 9 Million Java Developers Worldwide
  - ✓ #1 Choice for Developers
  - ✓ #1 Development Platform
  - ✓ 3 Billion Mobile Phones Run Java
  - ✓ 100% of Blu-ray Disc Players Ship with Java
  - ✓ 5 Billion Java Cards in Use
  - ✓ 125 million TV devices run Java
  - ✓ 5 of the Top 5 Original Equipment Manufacturers Ship Java ME

# Java is everywhere!



IEEE Spectrum   Trending   Web   Enterprise   Jobs   Open   Custom   Mobile   Embedded

Create custom ranking   (Click to hide)

## Language Ranking: IEEE Spectrum

Rank	Language	Type	Score
1	Python	Web   Desktop   Embedded	100.0
2	Java	Web   Mobile   Desktop	96.3
3	C	Mobile   Desktop   Embedded	94.4
4	C++	Mobile   Desktop   Embedded	87.5
5	R	Desktop	81.5
6	JavaScript	Web	79.4
7	C#	Web   Mobile   Desktop   Embedded	74.5
8	Matlab	Desktop	70.6
9	Swift	Mobile   Desktop	69.1
10	Go	Web   Desktop	68.0

IEEE Spectrum   Trending   Web   Enterprise   Jobs   Open   Custom   Mobile   Embedded

Create custom ranking   (Click to hide)

## Language Ranking: Jobs

Rank	Language	Type	Score
1	Python	Web   Desktop   Embedded	100.0
2	Java	Web   Mobile   Desktop	97.5
3	C	Mobile   Desktop   Embedded	96.0
4	C++	Mobile   Desktop   Embedded	85.7
5	JavaScript	Web	83.1
6	C#	Web   Mobile   Desktop   Embedded	77.2
7	HTML,CSS	Web	75.6

Source: [IEEE Spectrum](#)

# Java Features



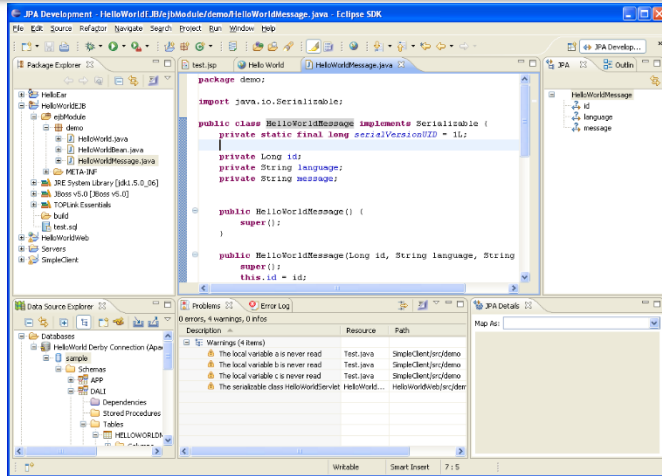
- Simple
  - No pointer
  - Automatic garbage collection
  - Not support multiple inheritances
  - Rich class library
- Object oriented:
  - All code is encapsulated in class.
  - All functions are associated with class.
  - Almost data types are objected

# Java Code Execution

Life Cycle



# Running Java-based Application



Java Code (.java)

JAVAC  
compiler

Byte Code (.class)

JVM

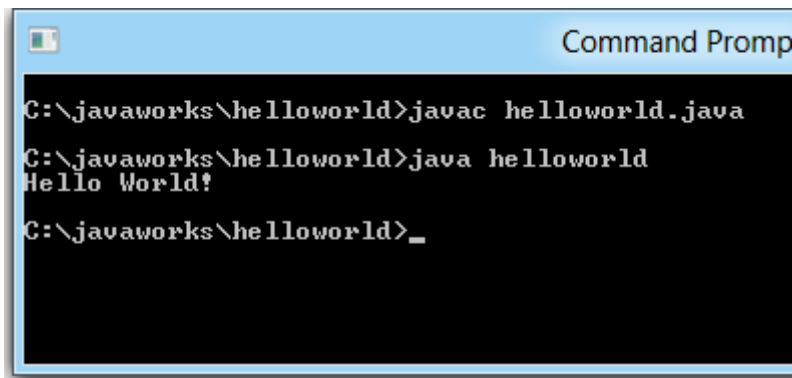
JVM

JVM

Windows

Linux

Mac



# From Source to Machine Code

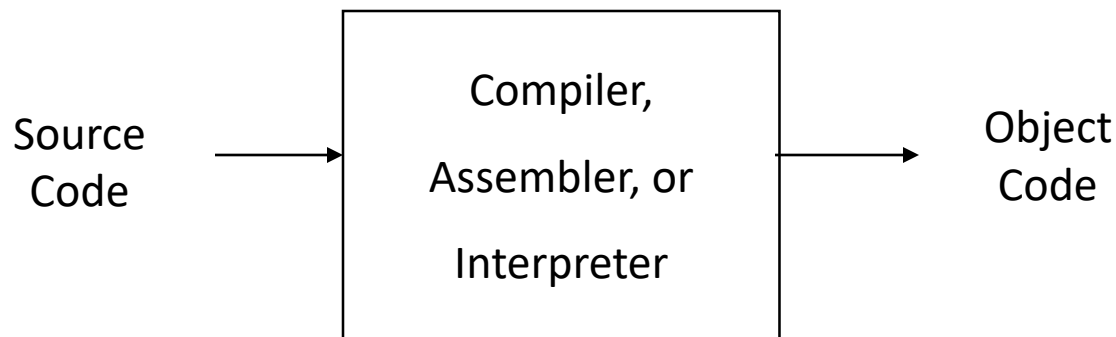


- *“Compiling a program”*  
translating from a high-level language (HLL) **source code to machine** (object, or executable) code.
- *“Compiler”*  
a program that translates HLL source code to machine (object, or executable) code.
- *“Assembly”*  
translating from **assemble language** source code **to machine** (object, or executable) code.
- *“Assembler”*  
a program that translates assembly source code to machine (object, or executable) code.
- Compilers need to know the specific target hardware

# Compilers / Assemblers / Interpreters



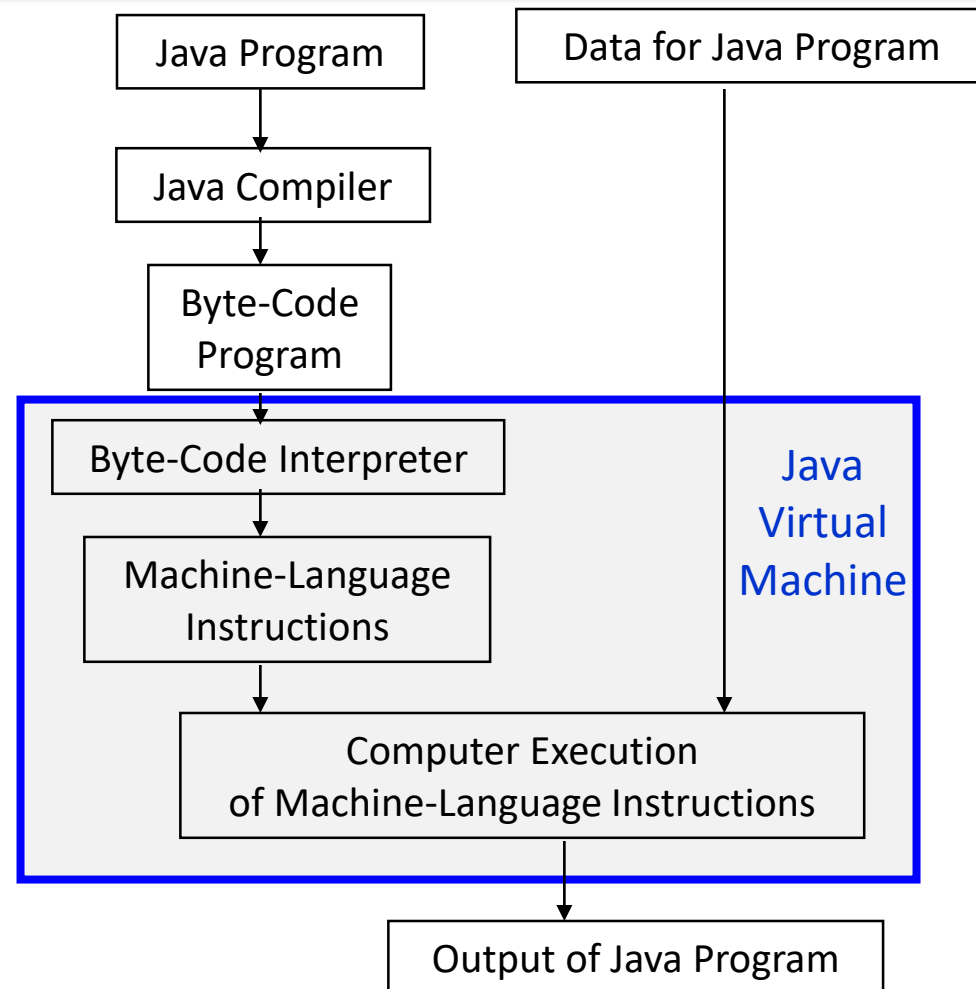
- Compilers and Assemblers
  - translation is a separate user step
  - translation is “**off-line**,” i.e. not at run time
- Interpreters - another way to translate source to object code
  - interpretation (from source to object code) is not a separate user step
  - translation is “**on-line**,” i.e. at run time



# Java Program Translation



- Both Compilation and Interpretation
- Intermediate Code:  
“*Byte Code*”
  - similar to assembly code, but hardware *independent*
- Interpreter translates from generic byte code to hardware-specific machine code





# Java Byte Code



- **Generated** by Java compiler
  - Instead of generating machine language as most compilers do, the Java compiler generates byte code.
- **Translated** to machine language of various kinds of computers
- **Executed** by Java interpreter
- Invisible to programmer
  - You don't have to know anything about how byte code works to write a Java program.



# Why Use Byte Code?



## Disadvantages:

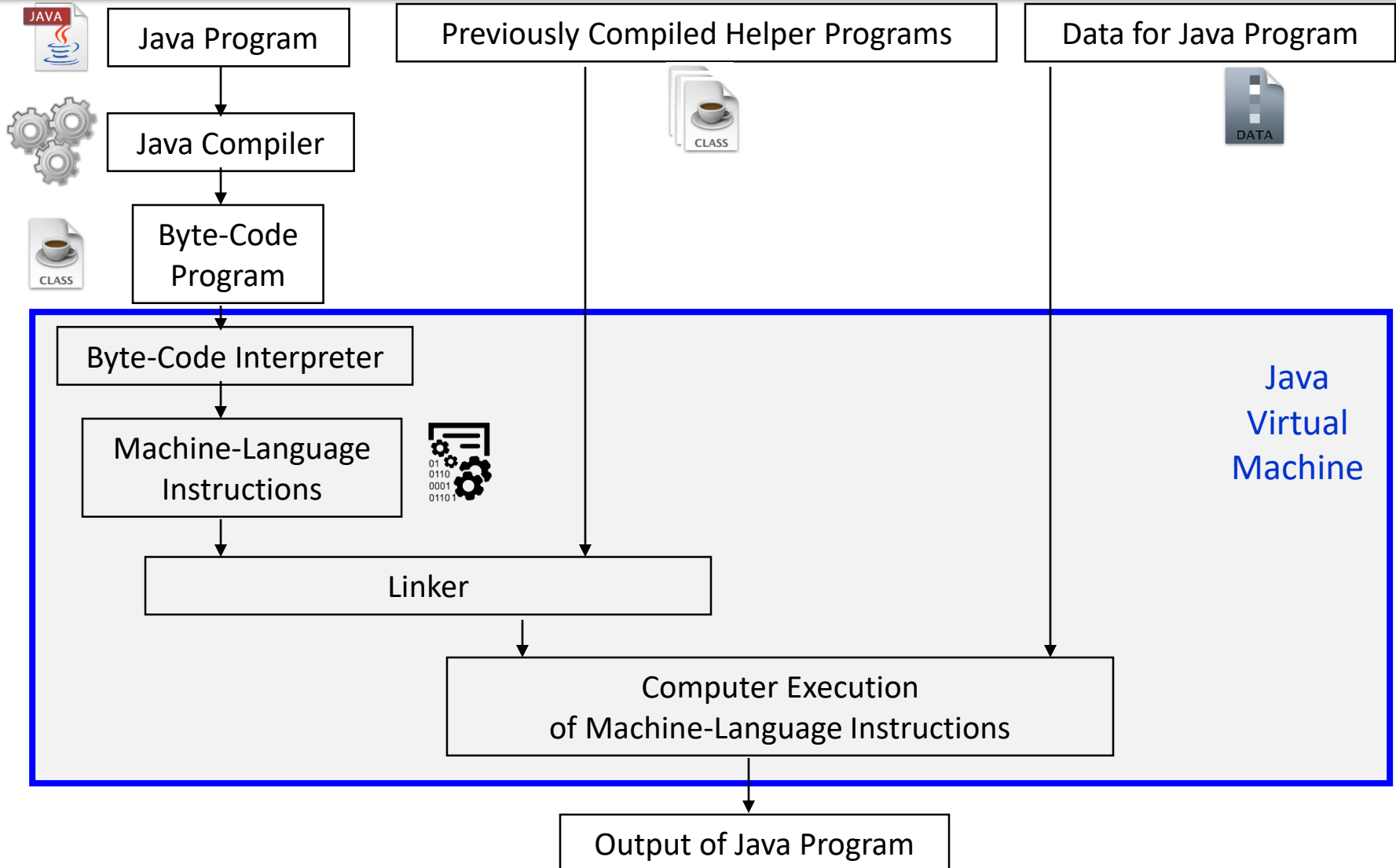
- requires both compiler and interpreter
- slower program execution



## Advantages:

- portability
  - very important
  - same program can run on computers of different types (useful with the Internet)
  - Java compiler for new types of computers can be made quickly

# Java Program Translation Including Linker

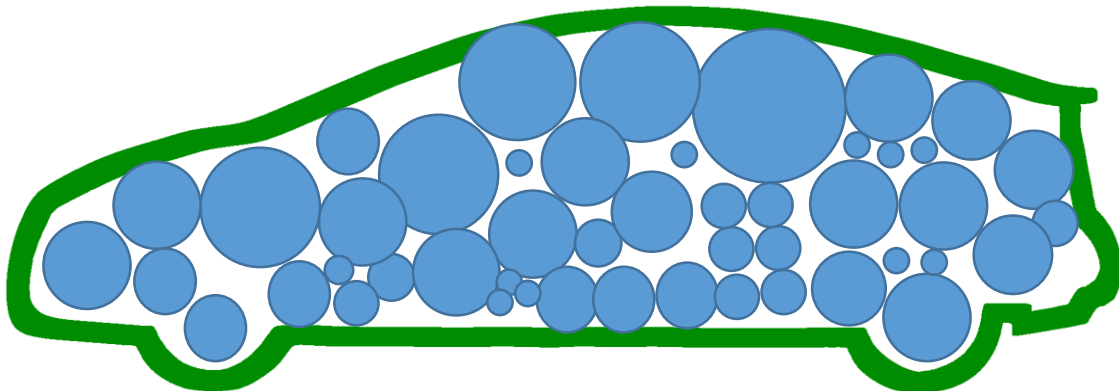


# Object-Oriented Programming (OOP)

# Objects and Methods



- Java is an object-oriented programming (OOP) language
  - Programming methodology that views a program as consisting of objects that interact with one another by means of actions (called **methods**)
  - Objects of the same kind are said to have the same type or be in the same class



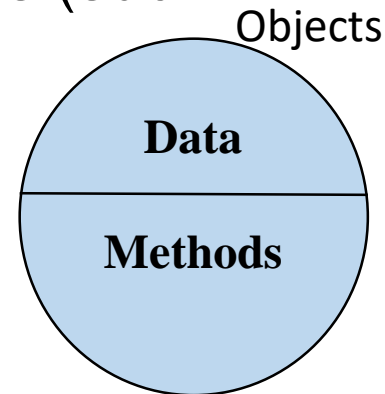
# Object-Oriented Programming: OOP



- A design and programming technique
- Some terminology:
  - **Object** - usually a person, place or thing (a noun)
  - **Method** - an action performed by an object (a verb)
  - **Type or Class** - a category of similar objects (such as automobiles)



- Objects have both **data** and **methods**
- Objects of the same class have the same data elements and methods
- Objects send and receive messages to invoke actions



# Example of an Object Class



# Example of an Object Class

## Class: Automobile

### Data Items:

- manufacturer's name
- model name
- year made
- color
- number of doors
- Body shape
- size of engine
- etc.

### Methods: (action)

- Define data items (specify manufacturer's name, model, year, etc.)
- Change a data item (color, engine, etc.)
- Display data items
- Calculate cost
- Assemble
- etc.



# Why OOP?



- Easy to design software as building blocks.
- Save development time (and cost) by reusing code
  - once a class is created, it can be used in other applications
- Easier debugging
  - classes can be tested independently
  - reused objects have already been tested

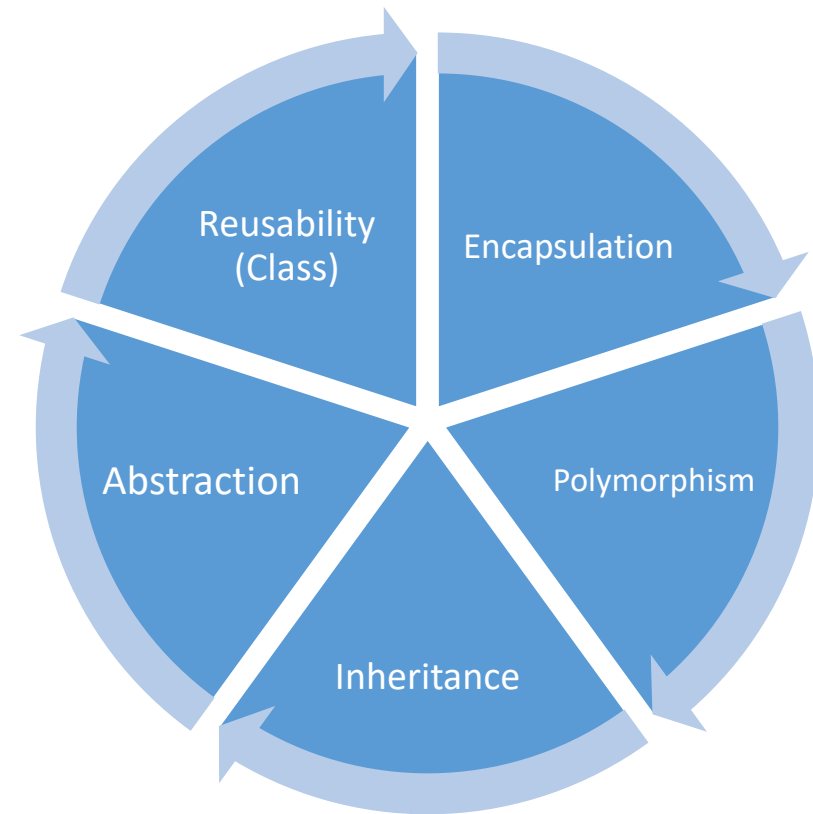


# Design Principles of OOP



Three main design principles of Object-Oriented Programming (OOP):

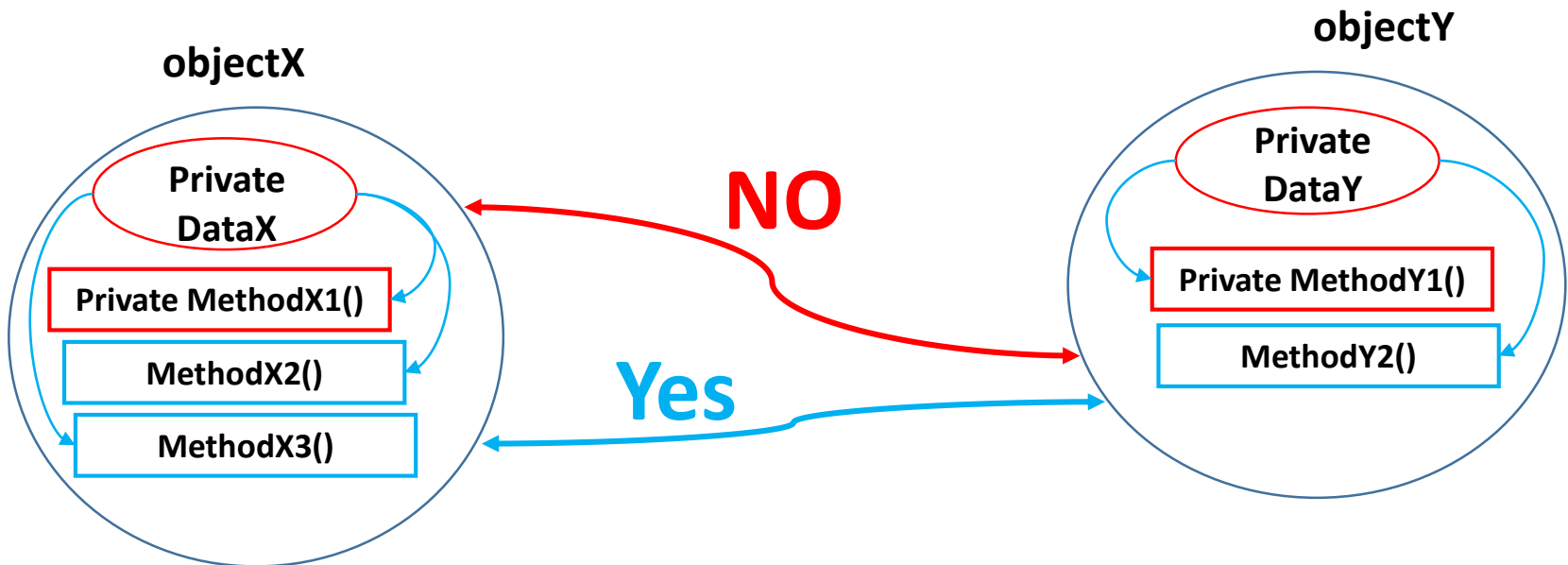
- Encapsulation
- Polymorphism
- Inheritance



# OOP: Encapsulation



- Design software
  - can be easily used
  - without knowing the details of how it works.
- Also known as *information hiding*



# OOP: Reusable Components



Advantages of using reusable components:

- saves time and money
- components that have been used before
  - often better tested and more reliable than new software

Make your classes reusable:

- encapsulation
- general classes have a better chance of being reused than ad hoc classes

# OOP: Polymorphism



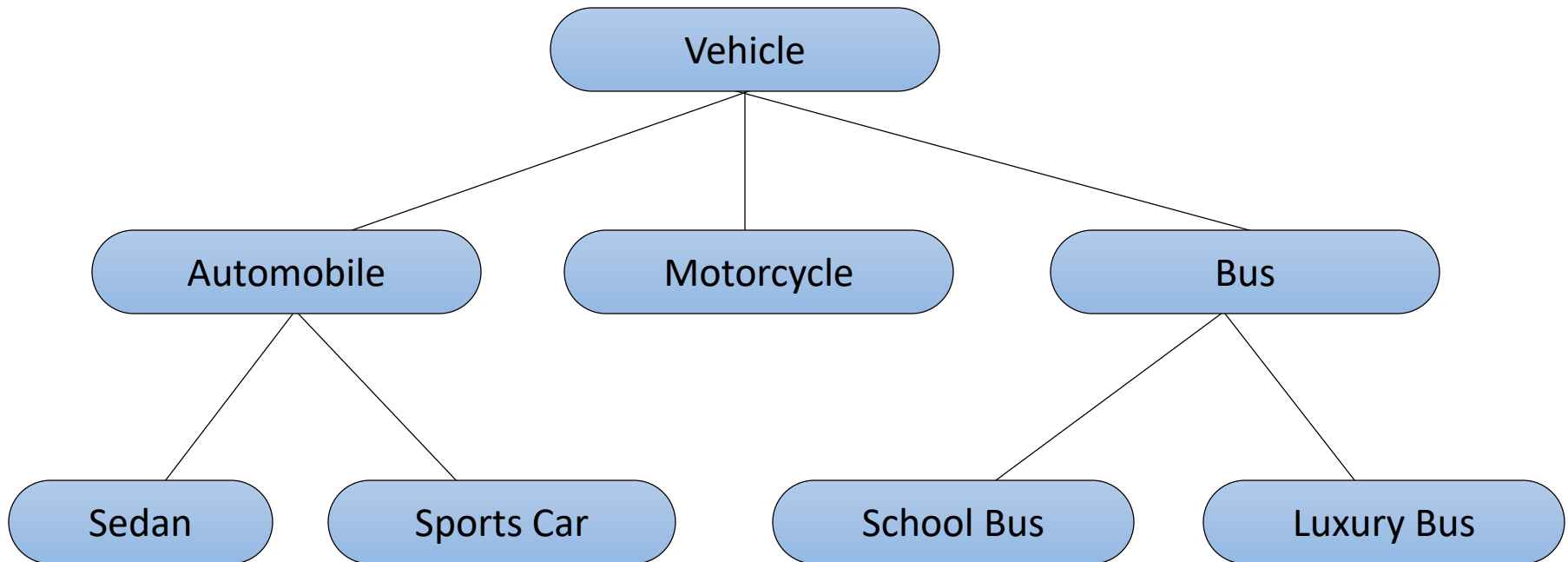
- Polymorphism—the same word or phrase can be mean different things in different contexts
- Analogy: in English, **bank** can mean:
  - side of a river or
  - a place to put money
- In Java, two or more classes could each have a method called **output**
- Each **output** method would do the “right thing” for the class that it was in. E.g.
  - display a number (Integer class)
  - display an image (Photo class)

# OOP: Inheritance



- Inheritance—a way of organizing classes
- Term comes from inheritance of traits like eye color, hair color, and so on.
- Classes with attributes in common can be grouped so that their common attributes are only defined once.

# An Inheritance Hierarchy



What properties does each vehicle inherit from the types of vehicles above it in the diagram?

# Java Simple Program



## Source

```
public class Program1
{
    public static void main(String[] arg)
    {
        System.out.println("Hello World");
    }
}
```

## Output

Hello World





# Programming Environment

Setup and practices

# Session Goals



- Install development tools
- Start using **Eclipse** editor software
- Practice **writing, compiling, and running** Java programs
- Gain familiarity with **syntax errors** and debugging

# Basic lab instructions

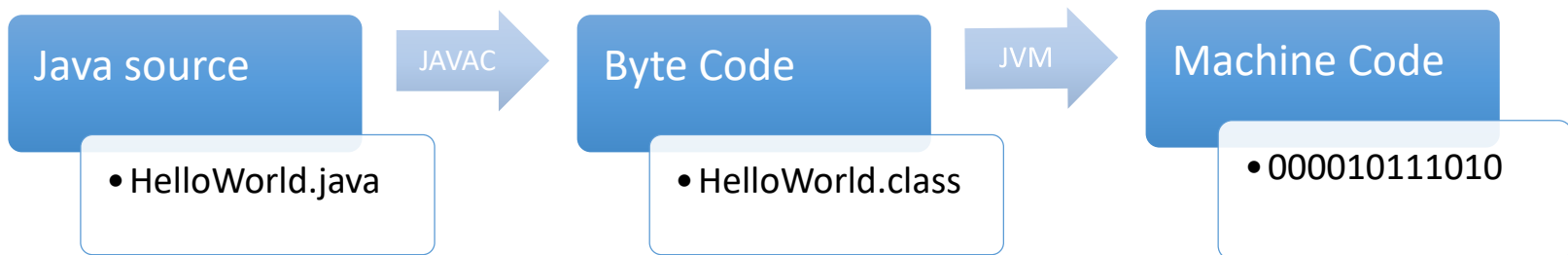


- Talk to your classmates for help. You can even work on the lab with a partner if you like.
- You may want to bring your textbook or look up the internet for syntax and examples.
- Stuck? Confused? Have a question? Ask a Lab Assistance (LA) for help, or look at the book or past lecture slides.
- Complete as many problems as you can within the allotted time. You don't need to keep working on these exercises after you leave the lab.
- Feel free to complete problems in any way.

# Recap Java



- Interpreted:
  - Compiled into byte code for the JVM (Java Virtual Machine).
  - Byte code is dependent on the Java platform, but is typically independent of operating system specific features.



# II. TUTORIAL



- ✓ Start project
- ✓ Your first program
- ✓ Running code
- ✓ Debug code
- ✓ Export to executable program

# 1. Software requirement

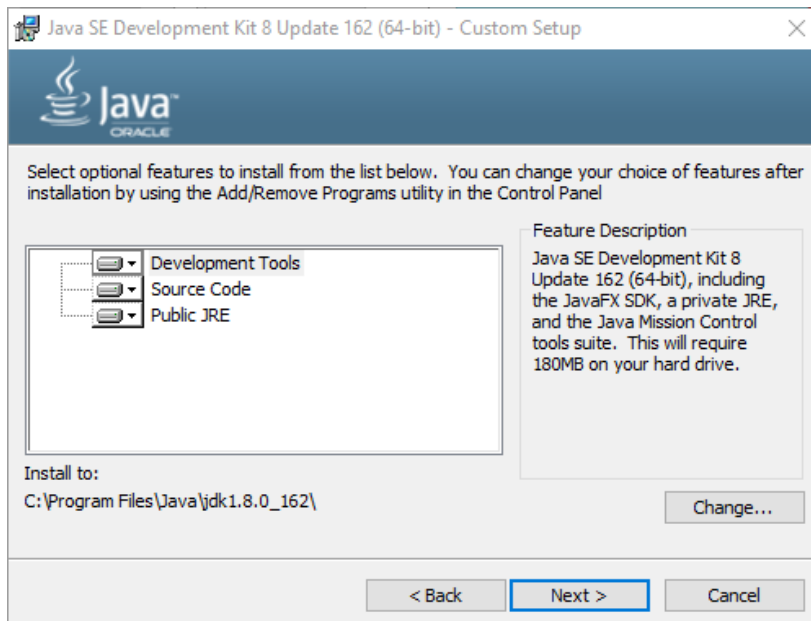


- Java SE Software Development Kit 8 (JDK 8~)

Link:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

- Run the execute file and follow the install steps.



# 1. Software requirement



- Eclipse for Java Developer  
Link:

<https://www.eclipse.org/downloads/packages/>



Try the Eclipse **Installer** 2019-12 R

The easiest way to install and update your Eclipse Development Environment.

[Find out more](#)

📥 1,959,487 Downloads

**Download**

Mac OS X 64 bit  
Windows 64 bit  
Linux 64 bit

### Eclipse IDE 2019-12 R Packages

Eclipse IDE for Enterprise Java Developers

353 MB 444,051 DOWNLOADS

Tools for Java developers creating Enterprise Java and Web applications, including a Java IDE, tools for Enterprise Java, JPA, JSF, Mylyn, Maven, Git and more.

[Click here to file a bug against Eclipse Web Tools Platform.](#)

[Click here to file a bug against Eclipse Platform.](#)

[Click here to file a bug against Maven integration for web projects.](#)

Windows 64-bit  
Mac Cocoa 64-bit  
Linux 64-bit

Eclipse IDE for Java Developers

201 MB 268,357 DOWNLOADS

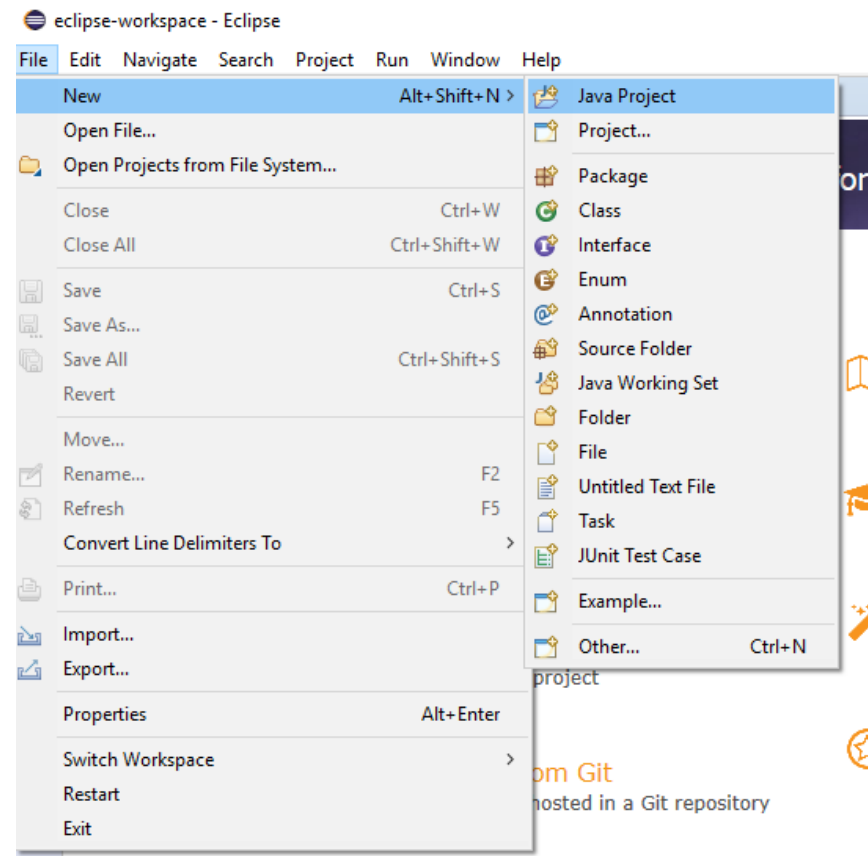
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration

Windows 64-bit  
Mac Cocoa 64-bit  
Linux 64-bit

# 2. Working with Eclipse

## Creating new project

- Open Eclipse, and create new project





# 2. Working with Eclipse

## Creating new project

**New Java Project**

Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location:  [Browse...](#)

**JRE**

☒ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ Use default JRE (currently 'jre1.8.0\_162') [Configure JREs...](#)

**Project layout**

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

**Working sets**

☐ Add project to working sets [New...](#)

Working sets:  [Select...](#)

[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

**New Java Project**

**Java Settings**

Define the Java build settings.

**Source** **Projects** **Libraries** **Order and Export**

**HelloJava**

- src

**Details**

[Create new source folder](#): use this if you want to add a new source folder to your project.

[Link additional source](#): use this if you have a folder in the file system that should be used as additional source folder.

[Add project 'HelloJava' to build path](#): Add the project to the build path if the project is the root of packages and source files. Entries on the build path are visible to the compiler and used for building.

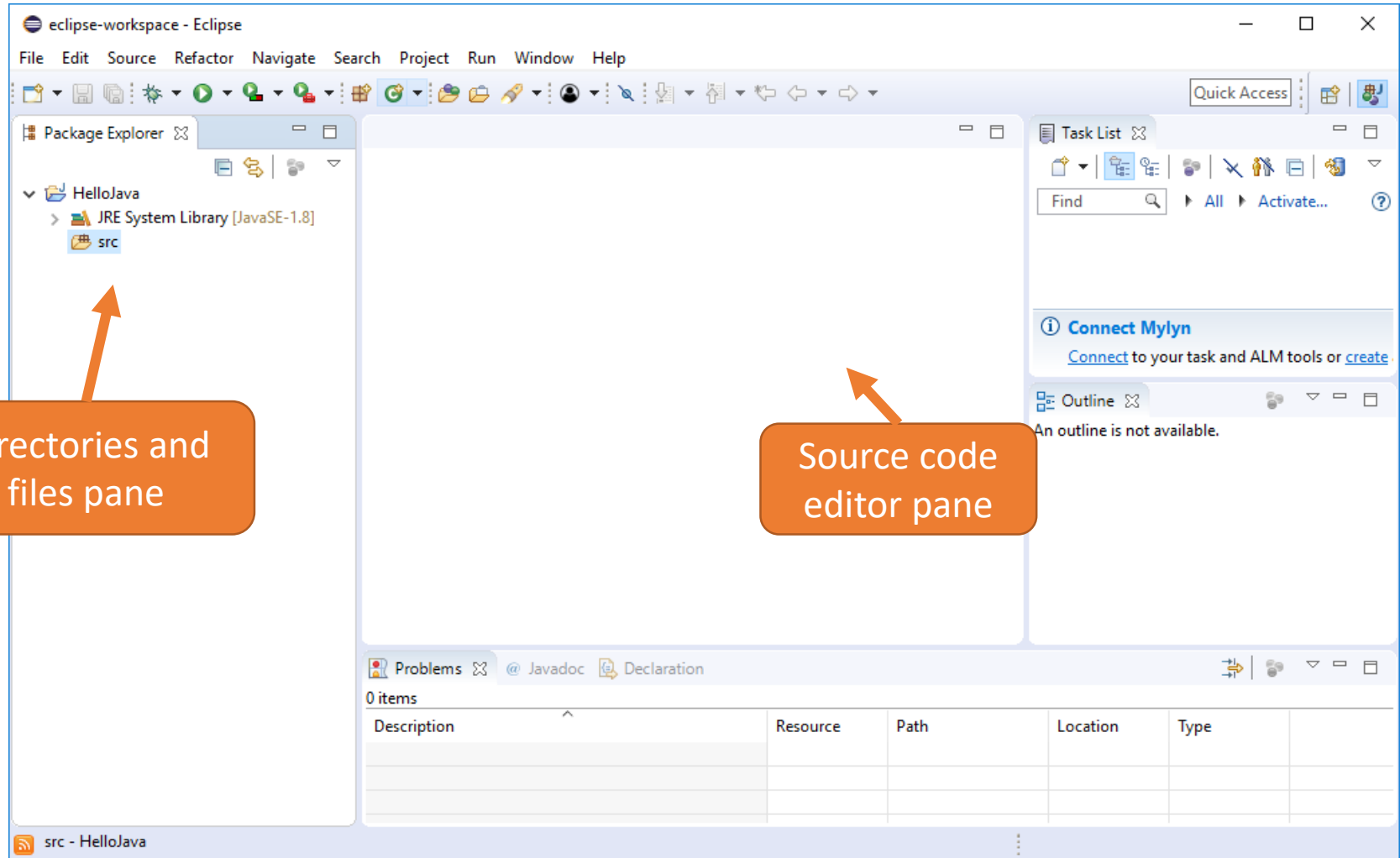
☐ Allow output folders for source folders

Default output folder:

[Browse...](#)

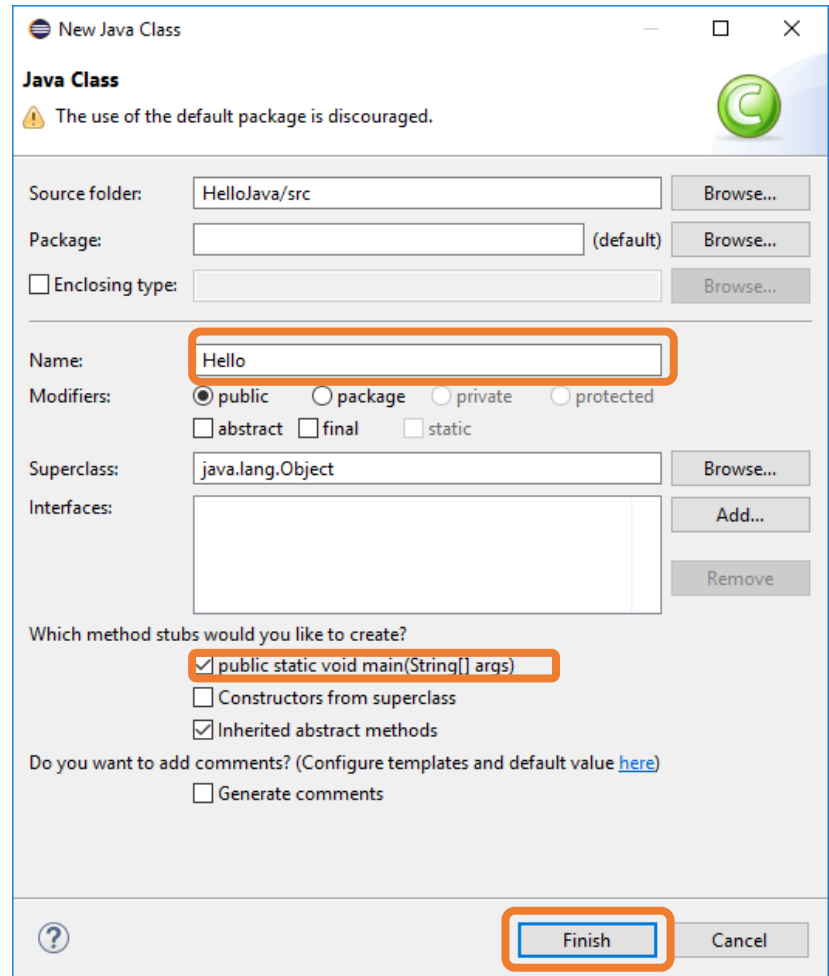
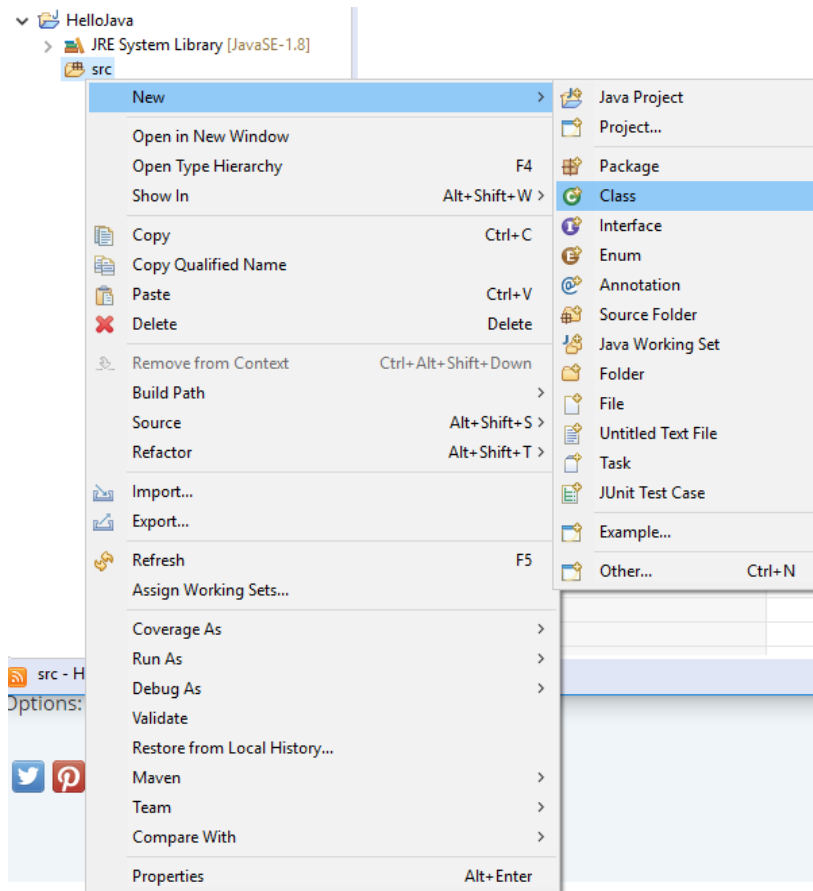
[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

# 2. Working with Eclipse Main GUI



# 3. First Application

## Creating a class



# 2. Working with Eclipse

## Creating a class



The screenshot shows the Eclipse IDE interface with the following components:

- Package Explorer:** Shows the project structure with 'HelloJava' containing 'src' and 'Hello.java'.
- Editor:** Displays the code for 'Hello.java':

```
1 public class Hello {  
2  
3  
4     public static void main(String[] args) {  
5         // TODO Auto-generated method stub  
6  
7     }  
8  
9 }  
10
```
- Task List:** Shows a search bar and 'All' tasks.
- Outline:** Shows the class 'Hello' and its method 'main(String[]): void'.
- Problems:** Shows '0 items'.
- Bottom Bar:** Includes 'Writable', 'Smart Insert', and '1:1'.

An orange callout box with the text "Some codes are generated" points to the TODO comment in the code.

# 3. First Application

## Printing output



- Using to print function in java console application
- Standard
  - System.out.print()
  - System.out.println()
- With format
  - System.out.printf()
  - System.out.format()

```
int a = 10;
int b = 20;
// Tedious string concatenation.
System.out.println("a: " + a + " b: " + b);
// Output using string formatting.
System.out.printf("a: %d b: %d\n", a, b);
```

# 3. First Application

## Printing output with format



- Format String:
  - **%** [**flags**] [**width**] [**.precision**] **conversion-character** (square brackets denote optional parameters )
- Flags:
  - **-** : left-justify ( default is to right-justify )
  - **+** : output a plus ( + ) or minus ( - ) sign for a numerical value
  - **0** : forces numerical values to be zero-padded ( default is blank padding )
  - **,** : comma grouping separator (for numbers > 1000)
  - : space will display a minus sign if the number is negative or a space if it is positive
- Conversion-Characters:
  - **d** : decimal integer [byte, short, int, long]
  - **f** : floating-point number [float, double]
  - **c** : character Capital C will uppercase the letter
  - **s** : String Capital S will uppercase all the letters in the string
  - **h** : hashCode A hashCode is like an address. This is useful for printing a reference
  - **n** : newline Platform specific newline character- use %n instead of \n for greater compatibility

# 3. First Application

## Printing output with format example



```
long n = 461012;
```

```
System.out.format("%d%n", n);           // --> "461012"
```

```
System.out.format("%08d%n", n);        // --> "00461012"
```

```
System.out.format("% ,8d%n", n);       // --> " 461,012"
```

```
System.out.format("%+,8d%n", n);       // --> "+461,012"
```

```
double pi = Math.PI;
```

```
System.out.format("%f%n", pi);         // --> "3.141593"
```

```
System.out.format("%.3f%n", pi);       // --> "3.142"
```

```
System.out.format("%10.3f%n", pi);     // --> "      3.142"
```

```
System.out.format("%-10.3f%n", pi);    // --> "3.142"
```

# 3. First Application

## Get the input



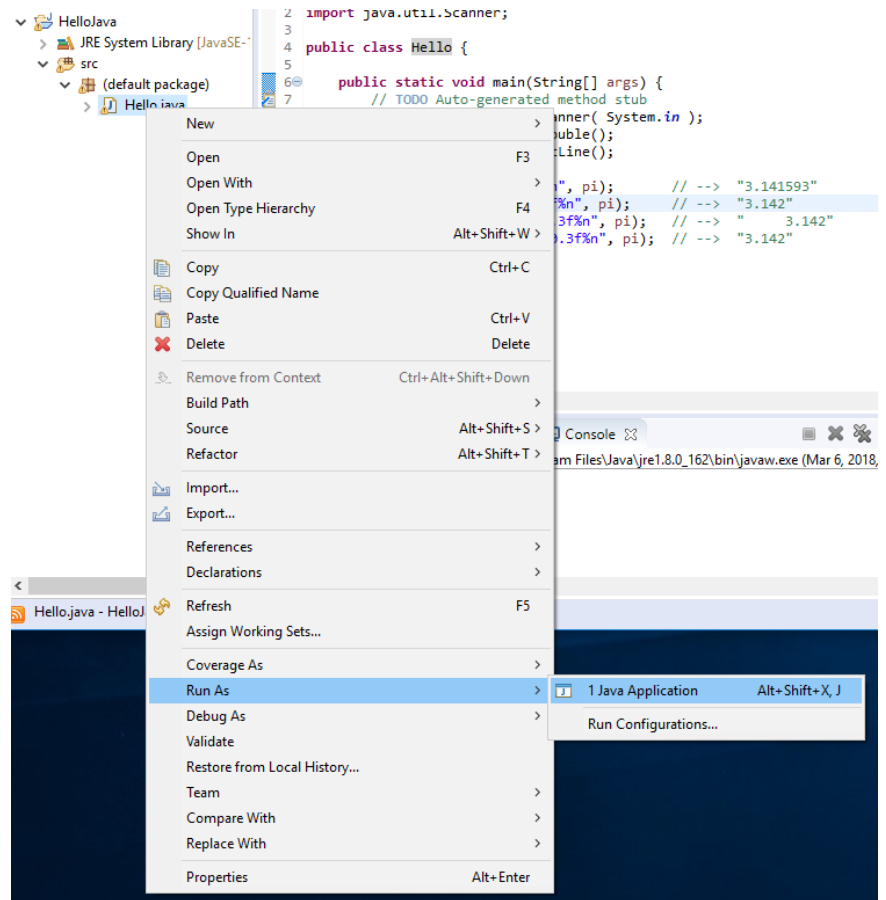
- Use Scanner object to read the input:
  - `Scanner input = new Scanner( System.in );`
- For different input type
  - `input.nextDouble()` to read in double
  - `input.nextInt()` to read in integer
  - `input.nextLine()` to read in a String

```
import java.util.Scanner;  
  
...  
// create Scanner to obtain input from command line  
Scanner input = new Scanner( System.in );  
int i= input.nextInt();  
String str = input.nextLine();
```



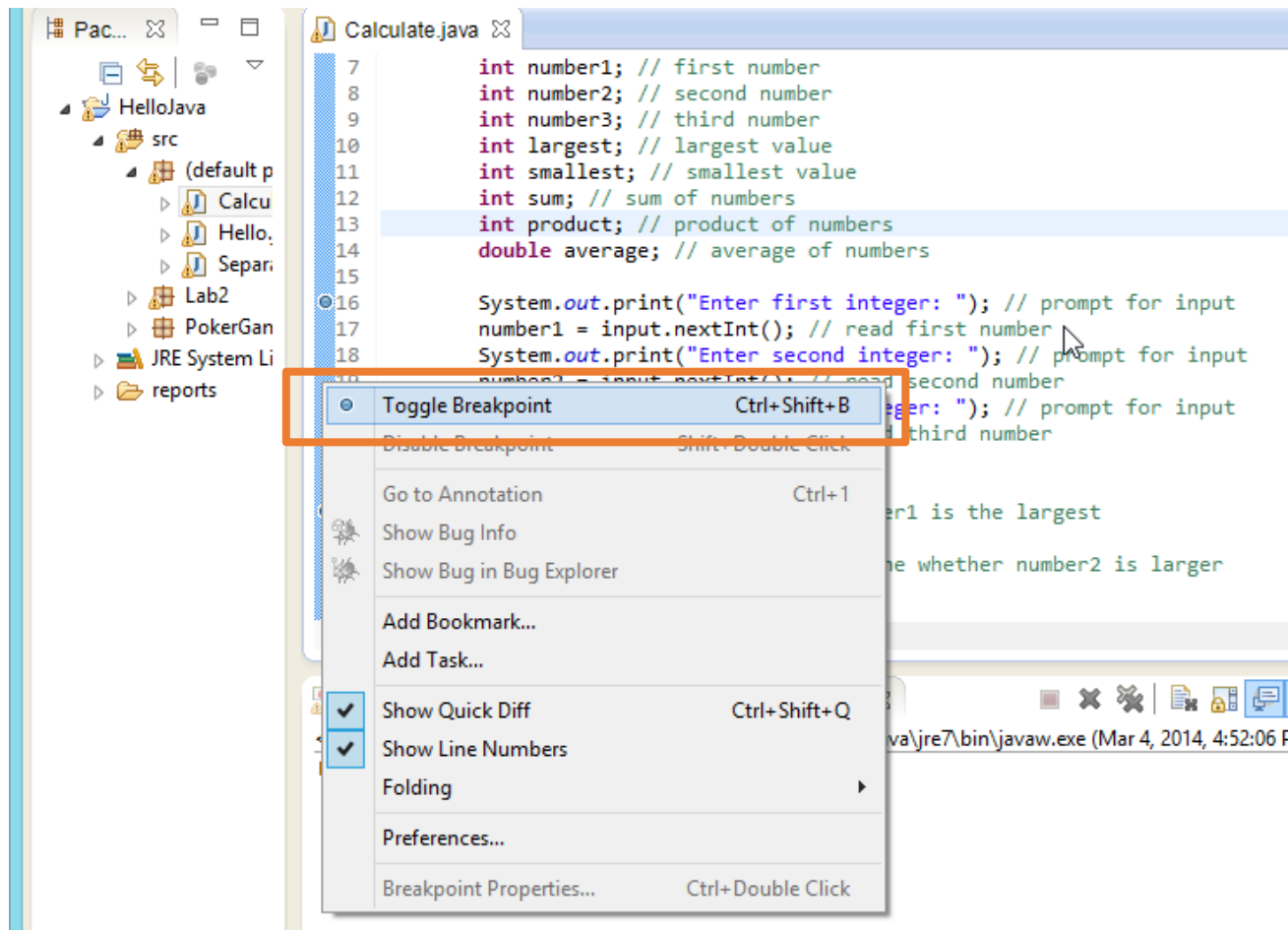
## 4. Running Your App

- Right click on the class  
Select Run As -> Java  
Application
- Menu Run -> Run
- Note:
  - Show console log:  
Menu Window->  
Show view ->  
Console



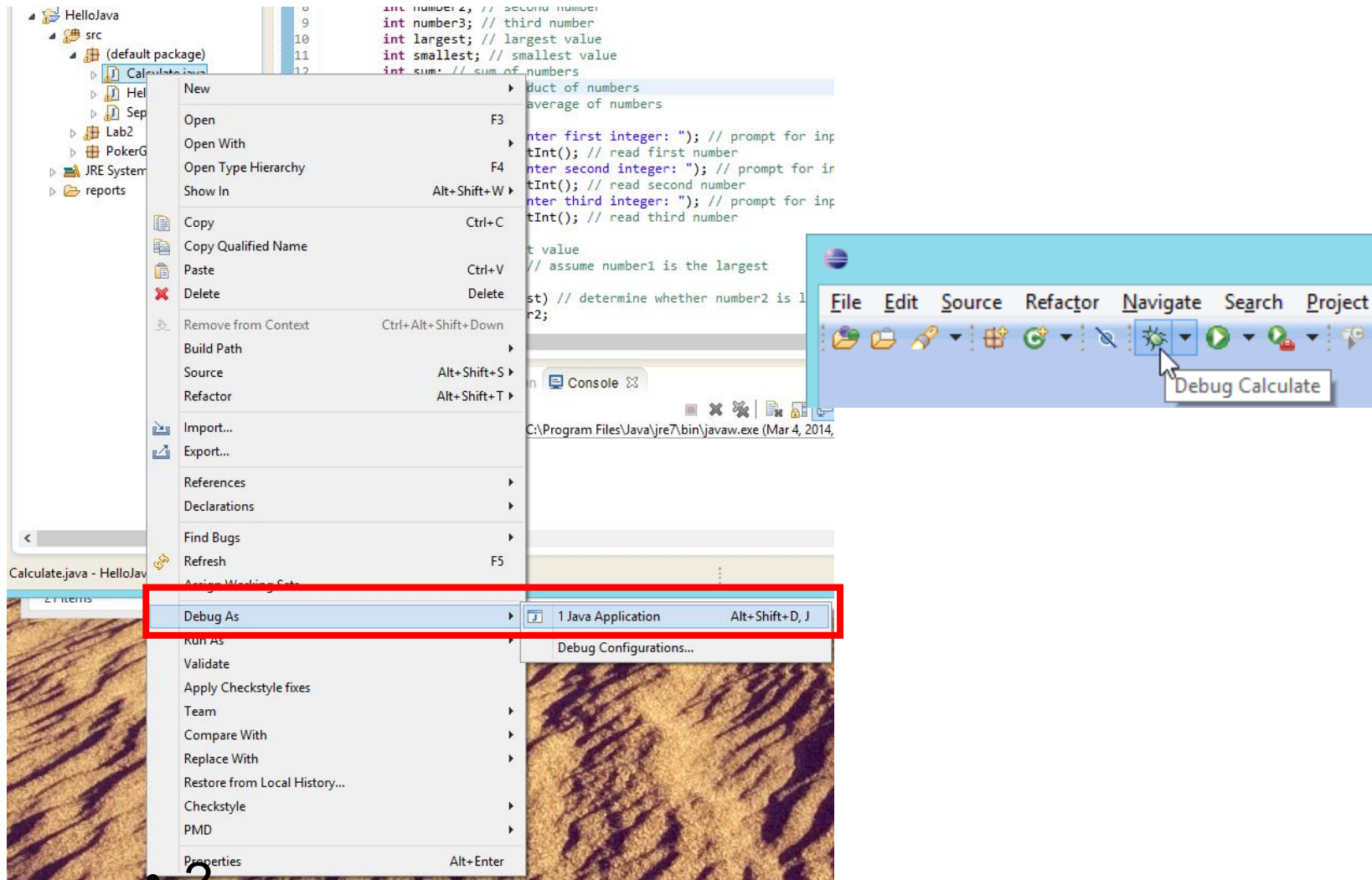
# 5. Debugging

## Setting breakpoints



# 5. Debugging

## Starting debugging



# 5. Debugging Debug perspective



**Confirm Perspective Switch**

? This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the state of the application.

Do you want to open this perspective?

☐ Remember my decision

Debug - HelloJava/src/Calculate.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access

Java PMD Debug

**Debug**

- Calculate [Java Application]
  - Calculate at localhost:50306
    - Thread [main] (Suspended (breakpoint at line 16 in Calculate))
      - Calculate.main(String[]) line: 16

C:\Program Files\Java\jre7\bin\javaw.exe (Mar 4, 2014, 5:07:35 PM)

**Variables**

Name	Value
args	String[] (id=16)
input	Scanner (id=18)

**Calculate.java**

```
7   int number1; // first number
8   int number2; // second number
9   int number3; // third number
10  int largest; // largest value
11  int smallest; // smallest value
12  int sum; // sum of numbers
13  int product; // product of numbers
14  double average; // average of numbers
15
16  System.out.print("Enter first integer: "); // prompt for input
17  number1 = input.nextInt(); // read first number
18  System.out.print("Enter second integer: "); // prompt for input
```

**Outline**

- Calculate
  - main(String[]) : void

**Console** **Tasks**

Calculate [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 4, 2014, 5:07:35 PM)

# Summary



- Overview about the course
- Introduction to Java
- Java Code Execution
  - How?
- Object-oriented programming
  - What?
  - Why?
- Development environment (Installation& Run)