# CSC3301: JAVA WORKSHOP


**Mohammed Hassan**

mhassan.se@buk.edu.ng

# CSC3301: JAVA WORKSHOP

## Administration

**Mohammed Hassan R1-04**

mhassan.se@buk.edu.ng

**Hafsa Kabir Ahmad R1-16**

hkahmad.cs@buk.edu.ng

**Mansur Babagana R3—59**

mbabagana.cs@buk.edu.ng

# Course Evaluation

Quizzes and Lectures --- 10%

Lab Exercise---20%

Written CA ---10%

Final Examination --- 60%

Total --- 100%

Pass Mark 45%

# Course outline

- Object and Classes (Chapter 1 and 2)
- Language Basics. (Chapter 3)
- Classes and Objects (Chapter 4)
- Inheritance (Chapter 5)
- Polymorphism (Chapter 7)
- Interfaces (Chapter 5)
- Packages (Chapter 5)
- Exceptions (Chapter 9)
- Input/Output (I/O) (Chapter 10)

# Week 1 & 2

- Object and Classes (Chapter 1 and 2)
  - Introduction to Objects
  - The Java Technology Phenomenon
  - The " Welcome to Java!" Application
  - What is an Object?
  - What is a Class?
- Language Basics. (Chapter 3)
  - Variables
  - Operators
  - Expressions, Statements, and Blocks
  - Control Flow Statements
- Running your first Java program

# Week 2 &3

- Classes and Objects (Chapter 4)
  - Classes
  - Constructors
  - Methods
  - Creating Objects
  - Using Objects
- Inheritance Part 1 (Chapter 5)
  - Inheritance: Key Definitions
  - The Purpose of Inheritance
  - An Example of Inheritance
  - What You Can Do in a Subclass
  - Private Members in a Superclass
  - Casting Objects
  - Overriding Methods
  - Hiding Fields
  - Constructor Chaining

# Week 3 & 4

- Inheritance Part 2  (Chapter 5)
  - Inheritance
    - IS-A versus HAS-A Relations
    - The Object Class as a Superclass
  - Debugging
    - Key Definitions
    - General strategy
    - jdb (a Command Line Debugger)

# Week 4 & 5

- Polymorphism Part 1 (Chapter 7)
  - Introduction
  - Examples
  - The Mechanics of Polymorphism
    - Static and Dynamic Binding
    - Casting Objects
  - The `instanceof` Operator
- Polymorphism Part 2
  - Examples
  - The Mechanics of Polymorphism
    - Abstract Classes (Chapter 5)
  - Private and Static Methods
  - Order of Constructor Calls
  - Designing Classes
  - Summary of Polymorphism

# Week 6

- Interfaces (Chapter 5)
  - What is an Interface?
  - Interfaces in Java
  - Interfaces and Multiple Inheritance
  - A Sample Interface, Relatable
  - Using an Interface as a Type
  - Rewriting Interfaces
  - Abstract Classes vs. Interfaces
  - Summary of Interfaces
- Packages (Chapter 5)
  - What are Packages
  - Creating and Using Packages
  - Naming Conventions
  - Using Package Members
  - Apparent Hierarchies of Packages
  - Managing Source and Class Files
  - Example
  - Summary of Creating and Using Packages

# Week 7 & 8

- Exceptions  Part 1 (Chapter 9)
  - Traditional Error Handling
  - Exceptions Vs. Traditional
  - What is an Exception?
  - Kinds of Exceptions
  - Exception Handling Keywords
  - Catching and Handling Exceptions
  - Keyword: finally
- Exceptions  Part 2
  - Exception Handling in Java
  - Catching Exceptions
  - The throws Clause
  - Statement 'throw'
  - Exception Chaining
  - Exception Handling
  - User Defined Exception Types
  - RuntimeException

# Week 9 & 10

- Input/Output (I/O) (Chapter 10)
  - Input/Output (I/O) Overview
  - Streams
    - Byte streams
    - Character streams
    - Buffered streams
    - Data streams
  - Summary

# Week 11

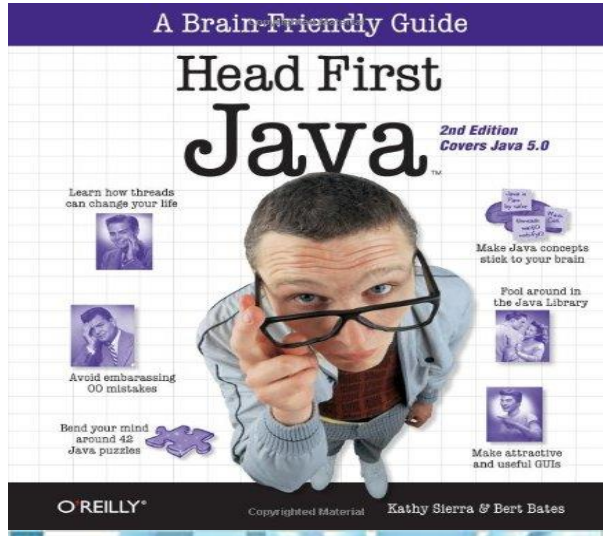
- Revision/Summary

# Week 12

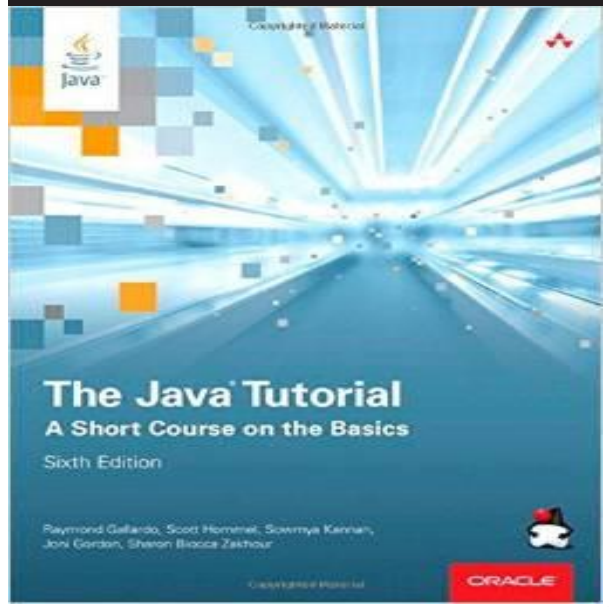
- Written and/or Lab CA

# Literature

- Head First Java, 2nd Edition, by Kathy Sierra and Bert Bates, O'REILLY, 2005.




- The Java Tutorial: A Short Course on the Basics (6th Edition) (Java Series) by R. Gallardo, et al., 2014.
  - Electronic version: http://docs.oracle.com/javase/tu

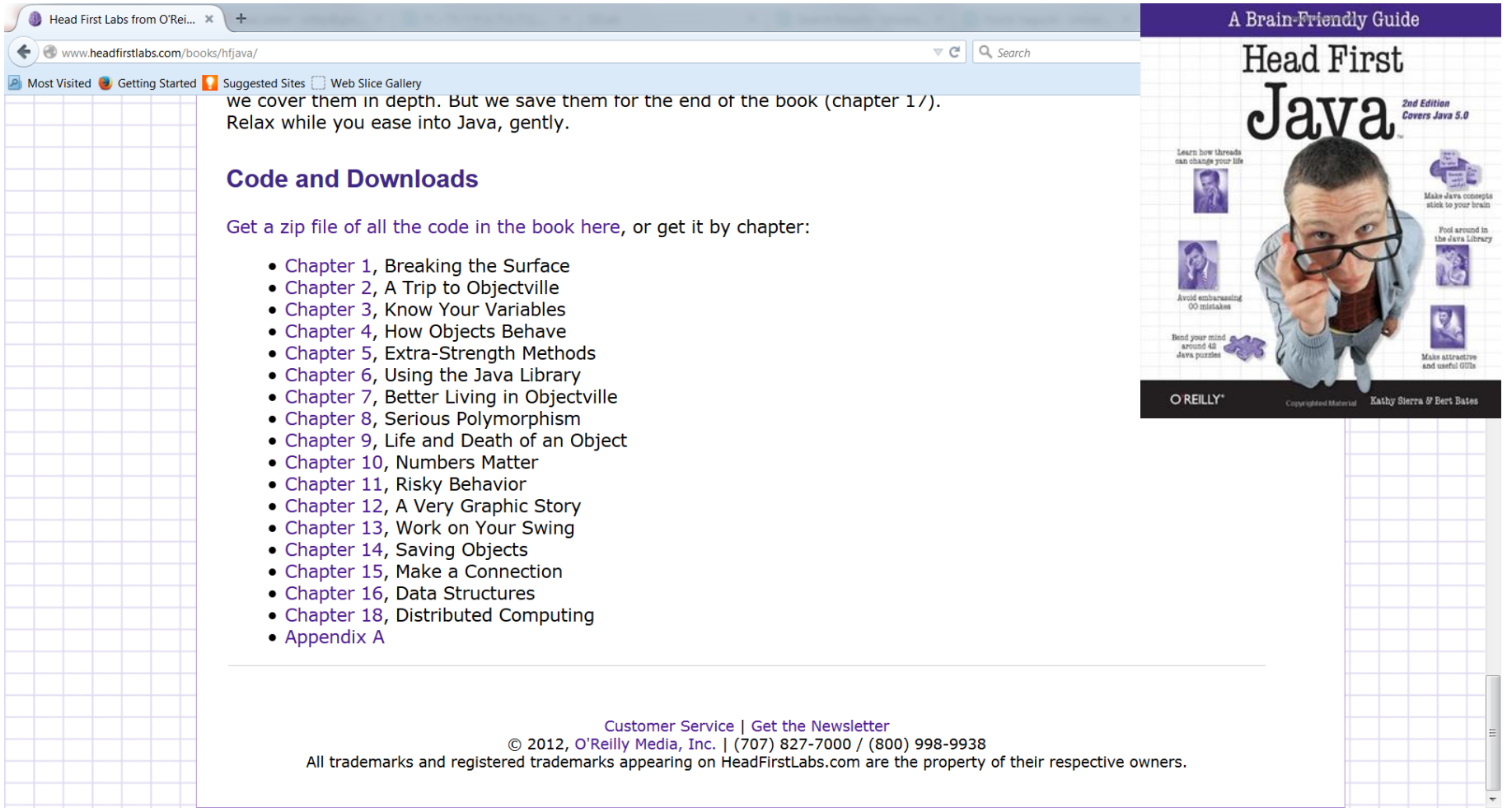


# Online Documentation

- Java provides online documentation for the whole environment:
  - How to compile and execute programs;
  - JDK (Java Development Kit) classes and their methods;
  - Many example programs;
  - Many documents that address different topics in Java.

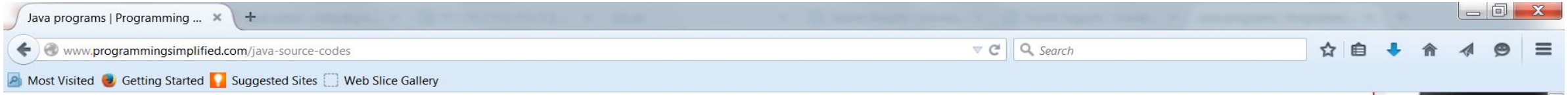
# Java Code Examples

http://www.headfirstlabs.com/books/hfjava/



# Java Code Examples

http://www.programmingsimplified.com/java-source-codes



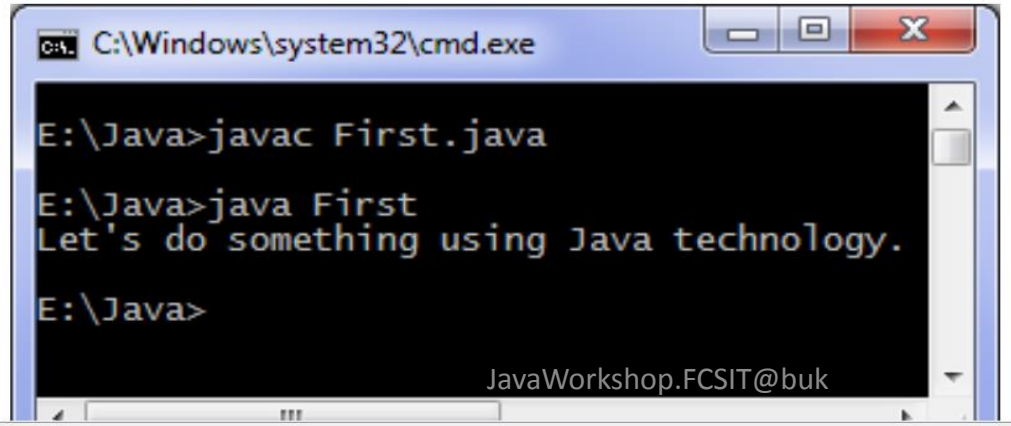

## Java programming examples

Example 1: Display message on computer screen.

```
class First {
  public static void main(String[] arguments) {
    System.out.println("Let's do something using Java technology.");
  }
}
```

This is similar to hello world java program. Download java programming class file.

Output of program:

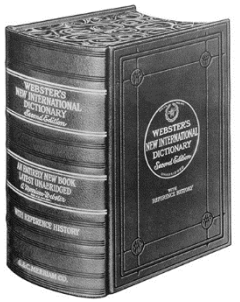


# How to study course materials

1. Read lecture slides (understand ALL examples);
2. Read the corresponding chapters in the text book.
3. Work on exercises.

# How to work on exercises

- Read problem descriptions very carefully: You MUST read ALL sentences.



- Go back to the lecture examples to find the analogies between them and the problem you should solve:
  - Make the bridges between them in your brain.

- When code is prepared, analyze the results produced by your code to make sure that they are correct.
- Following these instructions, you become INDEPENDENT!

# Requirements for the Lectures

- Pay attention to the lecture and ask as much questions as possible: *reasonable questions please!*
- There will be a quiz at EVERY lecture (to answer the questions, you have to follow the lecture)
- Quiz sheets will be available ONLY at the lecture time. It is not allowed to get the quiz questions after the lecture.
- Quiz sheets will be collected at the end of the class: One student can submit only one sheet.

# How to get credits for the exercises

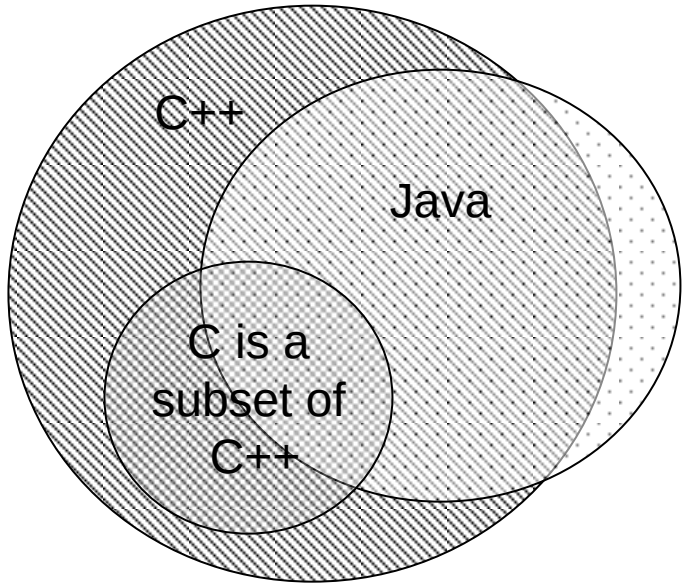
- To get credits for your solution, you should
  - show your TA or your instructor how your programs work;
  - answer their questions;
  - submit your source code to your instructor.

# Exercise grading policy

- The penalty for the late submissions is 0% of points for each problem;
- Late submissions are NOT acceptable after the practical hours;

# Contents for the today

- The Java Technology Phenomenon
- The " Welcome to Java!" Application
- What is an Object?
- What is a Class?



# The Java Technology Phenomenon

- Java technology
  - The Java programming language
  - The Java platform
- The Java programming language is a high-level language that can be characterized:

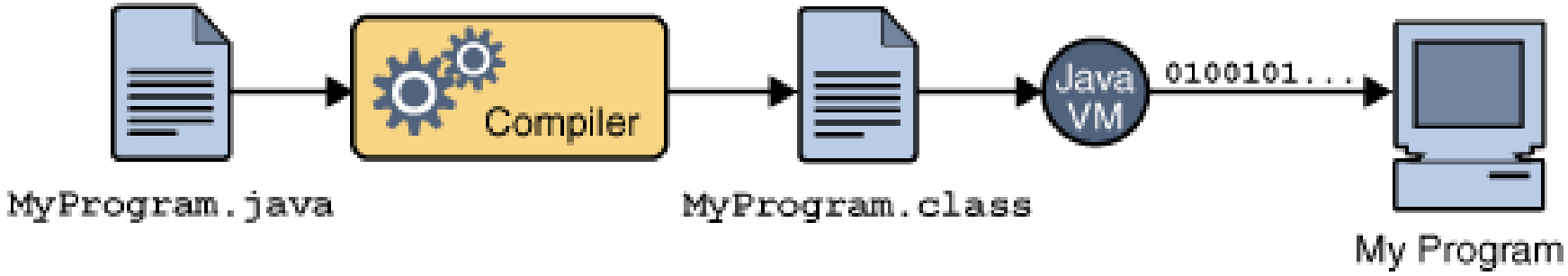
| | |
|---|---|
| Simple | Architecture neutral |
| Object oriented | Portable |
| Distributed | High performance |
| Multithreaded | Robust |
| Dynamic | Secure |

Each of these words is explained in *The Java Language Environment*

http://www.oracle.com/technetwork/java/index-136113.html

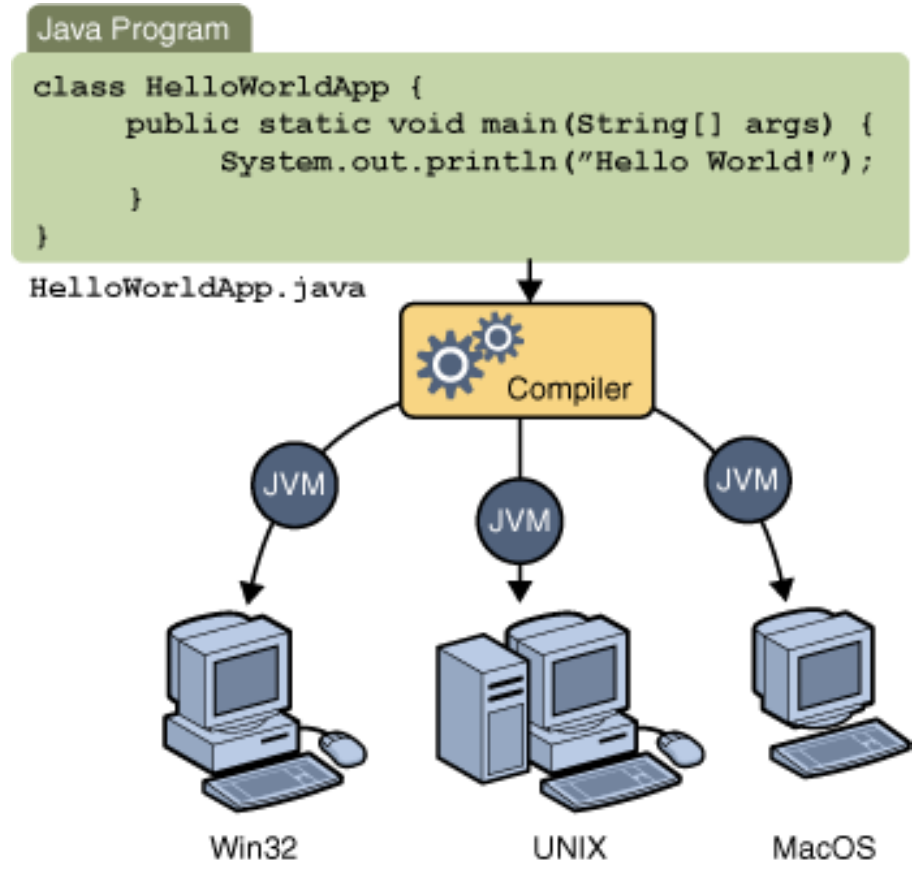
# The Java programming language

- All source code is first written in plain text files ending with the *.java* extension.
- Those source files are then compiled into .class files by the *javac* compiler. A *.class* file does not contain code that is native to your processor; it contains *bytecodes* — the machine language of the Java Virtual Machine (Java VM).
- The java launcher tool then runs your application with an instance of the Java Virtual Machine.



# The Java VM

- The Java VM is available on many different operating systems.
- The same *.class* files may run on
  - Microsoft Windows,
  - the Solaris ™ Operating System (Solaris OS),
  - Linux, or
  - Mac OS.



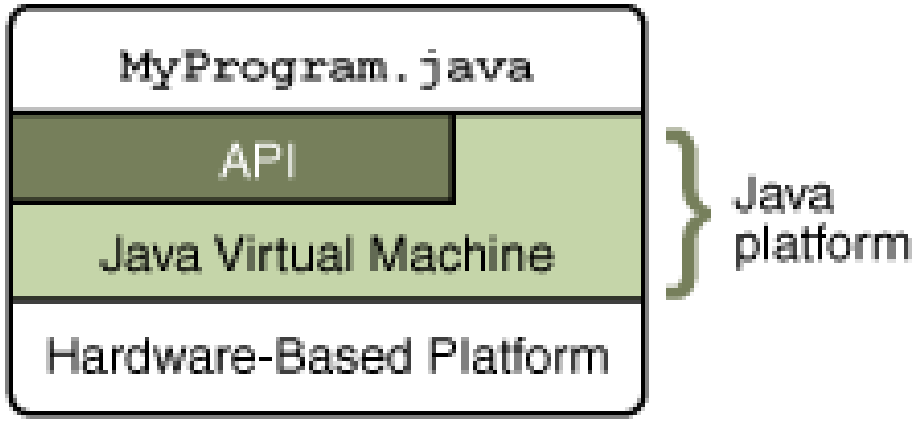
# The Java Platform

- A *platform* is the hardware or software environment in which a program runs.
  - The most popular platforms:
    - Microsoft Windows,
    - Linux,
    - Solaris OS, and
    - Mac OS.
- Most platforms can be described as a combination of the operating system and underlying hardware.

# The Java Platform

- The *Java platform* differs from most other platforms in that it's a *software-only platform* that runs on top of other hardware-based platforms.
- The Java platform has two components:
  - The *Java Virtual Machine*
  - The *Java Application Programming Interface* (API)
- The API is a large collection of ready-made software components.
- The API provides many useful capabilities:
  - It is grouped into libraries of related classes and interfaces;
  - These libraries are known as *packages*.

# The Java Platform




- The API and Java Virtual Machine insulate (protect) the program from the underlying hardware.
- As a platform-independent environment, the Java platform can be **a bit slower** than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without influencing portability.

# What Can Java Technology Do?

- **Development Tools**: They  provide everything you need for compiling, running, monitoring, debugging, and documenting your applications.  The main tools are:
  - The compiler: *javac*
  - The launcher: *java*
  - the documentation tool: *javadoc*.
- **Application Programming Interface (API)**: The API provides the core functionality of the Java language:
  - It offers a wide array of useful classes ready for use in your own applications.
  - It spans everything from basic objects, to networking and security, to XML generation and database access, and more.

# What Can Java Technology Do?

- **Deployment Technologies**: The JDK software provides standard mechanisms such as the Java Web Start software and Java Plug-In software for deploying your applications to end users.
- **User Interface Toolkits**: The Swing and Java 2D toolkits make it possible to create sophisticated Graphical User Interfaces (GUIs).
- **Integration Libraries**: Integration libraries such as the Java IDL API, JDBC™ API, Java Naming and Directory Interface™ ("J.N.D.I.") API, Java RMI, and Java Remote Method Invocation over Internet Inter-ORB Protocol Technology (Java RMI-IIOP Technology) enable database access and manipulation of remote objects.

# The " Welcome to Java!" Application

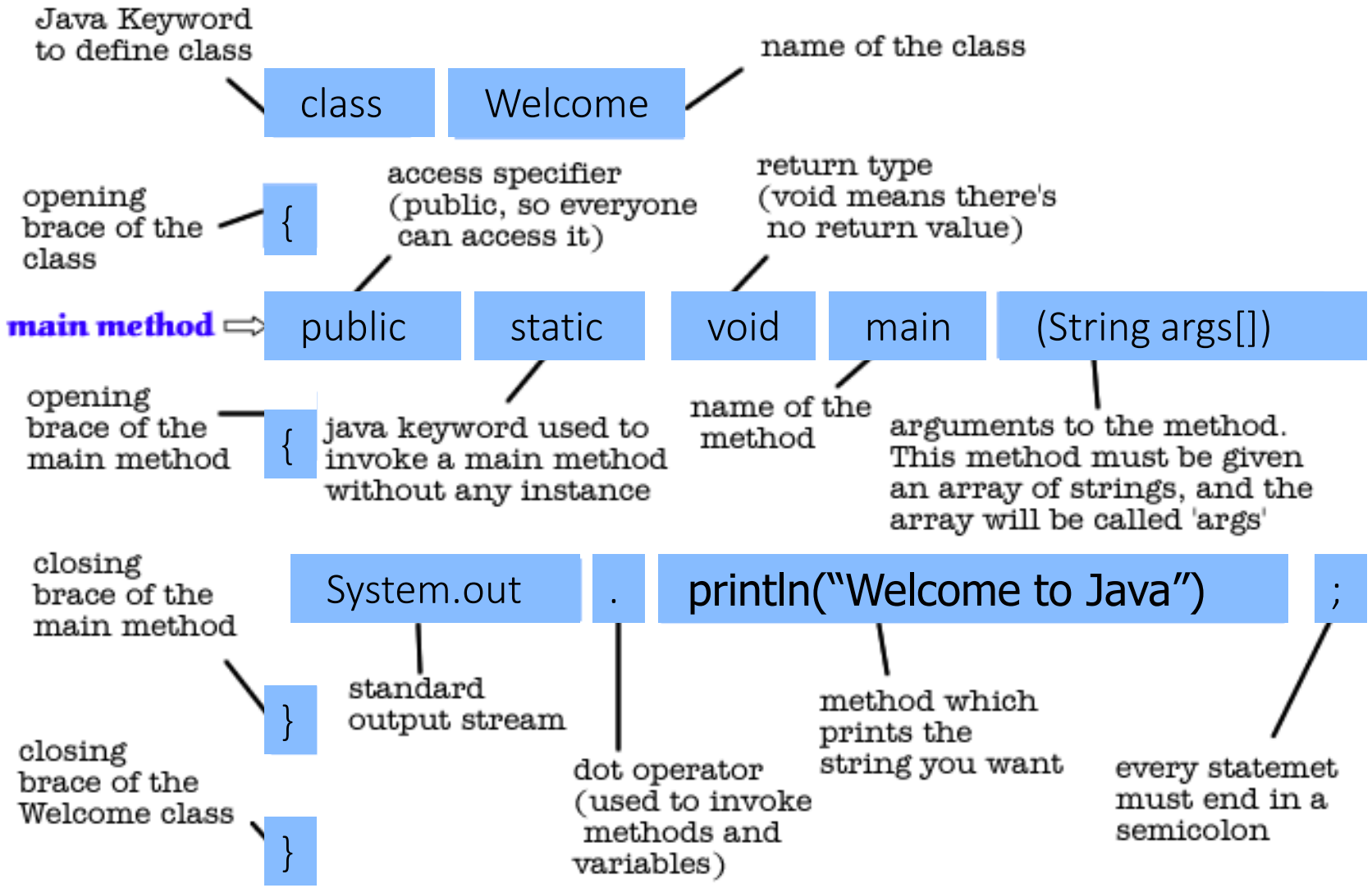
```
/**
 * The Welcome class
 * implements an application that
 * simply displays “Welcome to Java!"
 * to the standard output.
 */
class Welcome {
    public static void main(String[] args) {
        System.out.println(“Welcome to Java!");
  //Display the string.
    }
}
```

- How to create, compile and run:
  - Start your favorite text editor (e.g., notepad, notepad++, emacs, etc).
  - Type the code (left side of the slide).
  - Save the text to the file: Welcome.java
  - Compile the program typing: javac Welcome.java
  - Run the program typing: java Welcome
  - Output of the program: Welcome to Java!

# A Closer Look at the "Welcome to Java!" Application

- The "Welcome to Java" application consists of three primary components:
  - source code comments,
  - the Welcome class definition, and
  - the main method.
- In Java, every application must contain a main method whose signature is:
  - *public static void main(String[] args)*
- The main method is similar to the main function in C and C++; it's the entry point for your application and will subsequently invoke all the other methods required by your program.

# A Closer Look at the "Welcome to Java!" Application



# What is an Object?

- Objects are key to understanding object-oriented technology.
- Real world objects are around us (e.g., dogs, bicycles, cars, houses, tables, people).
- Objects share 2 characteristics:
  - State – the data of interest (e.g., people have a name, hair color, date of birth, etc.)
  - Behavior – what objects do (e.g., people walk, eat, read, etc.)

# Example

- Example: Dogs have
  - state (or properties)
    - name, color, eye color, height, length, weight.
  - behavior (or methods)
    - barking, fetching, sitting, laying down, wagging tail.

# Software Objects

- Software objects are conceptually similar to real-world objects: They consist of state and related behavior.
  - An object stores its state in *fields* (variables in some programming languages).
  - An object exposes its behavior through *methods* (functions in some programming languages). Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication.

# What is Encapsulation?

- Hiding internal state and requiring all interaction to be performed through an object's methods is known as *data encapsulation* — a fundamental principle of object-oriented programming.
  - In other words, there is not direct access to the fields of the object.

# Software Objects: Benefits

- Modularity
  - The source code for an object can be written and maintained independently of the source code for other objects.
- Information-hiding
  - By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
- Code re-use
  - If an object already exists (perhaps written by another software developer), you can use that object in your program.
- Pluggability and debugging ease
  - If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace *it*, not the entire machine.

# What Is a Class?

- In the real world, there are many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model.
- Each bicycle was built from the same set of blueprints (detailed plan, pattern) and therefore contains the same components.
- In object-oriented terms, we say that your bicycle is an *instance* of the *class of objects* known as bicycles.

Class Bicycle

Two objects of the Bicycle class

# What Is a Class?

- A *class* is the blueprint (detailed plan, template)  from which individual objects are created.
- It defines the characteristics and behaviors of all objects of a certain type.
- In other words, a class is an abstraction that contains the attributes and behaviors common to all objects of a given type.

# Implementation of a Bicycle

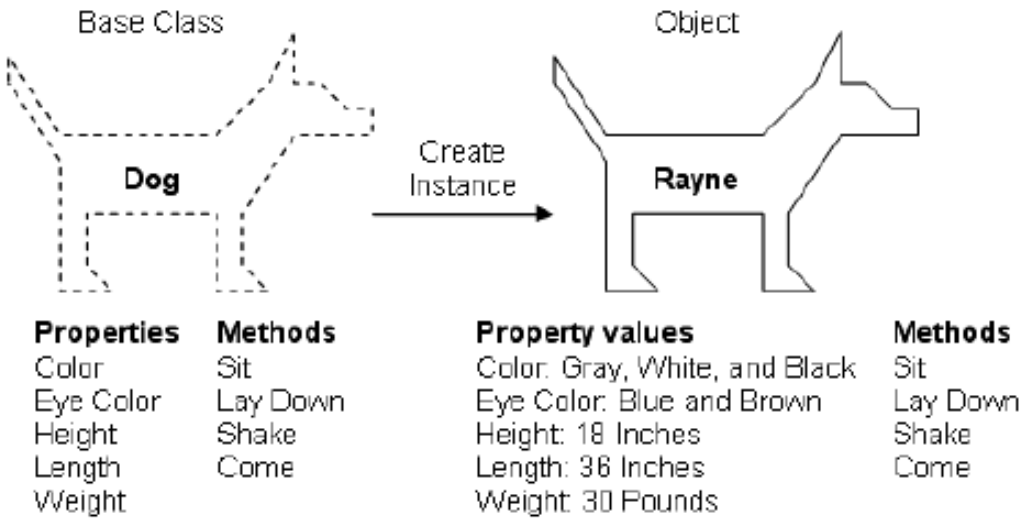
```
// File: BicycleDemo.java
class Bicycle {          // fields or attributes
    int cadence = 0;
    int speed = 0;
    int gear = 1;
  Bicycle()  {           // constructor
          cadence = 0;
          speed = 0;
          gear = 1;
  }                      // methods
    void changeCadence(int newValue) {
        cadence = newValue;
    }
    void changeGear(int newValue) {
        gear = newValue;
    }
    void speedUp(int increment) {
        speed = speed + increment;
    }
    void applyBrakes(int decrement) {
        speed = speed - decrement;
    }
    void printStates() { System.out.println( "cadence:“
  +  cadence+ " speed:"+speed+" gear:"+gear);
    }
} // end of the Bicycle class
class BicycleDemo {
    public static void main(String[] args) {
        // Create two different Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();
      bike1.speedUp(10);  // Invoke methods on those
      bike1.printStates();   // objects
        bike2.changeCadence(50);
        bike2.speedUp(15);
      bike2.printStates();
    }
}
```

# Comments on the Previous Slide

- The fields cadence (the rate at which a cyclist is turning the pedals.), speed, and gear (rate at which the rider's legs turn compared to the rate at which the wheels turn) represent the object's state, and the methods (changeCadence, changeGear, speedUp etc.) define its interaction with the outside world.
- The responsibility of creating and using new Bicycle objects belongs to the BicycleDemo class.
- Compile
  - javac  BicycleDemo.java
- Run:
  - java BicycleDemo
- Output:
  - cadence:0 speed:10 gear:1
  - cadence:50 speed:15 gear:1

# What is the Difference between a Class and an Object?

- One class – many objects: The class tells the Java virtual machine how to make an object of that particular type. Each object made from that class can have its own values for  the instance variables.




# Procedural versus Object-Oriented

DATA

```
typedef struct people {          // type
    char* name;
    char hairColor[32];
    char dateOfBirth[128];

} People;
```

PROCEDURES

```
People* createPeople(char* name, …)


void eat(People* people)
void read(People* people)
void walk(People* people)
```

DATA & PROCEDURES: *encapsulation*

```
class People {          // type
    private String name;
    private String hairColor;
    private String dateOfBirth;




                                // constructor
    People(String name,String color,…)

                                // methods
    public void eat()
    public void read()
    public walk()

}
```

# Comments on the Previous Slide

- Procedural approach:
  - Functions are introduced to reduce the size of the programs, improve readability in them, and simplify the debugging process of large programs.
  - The original data may easily get corrupted:
    - The data are accessible to all the functions, even to those which do not have any right to access them.
- Object-oriented approach:
  - The data and the functions, which are supposed to have the access to the data, are put into one box known as an object.
  - There are no chances of any unauthorized access to the data.
  - See slide: Software Objects: Benefits (Sl. 18).

# Lab Exercise:
# Problem one

Study the *Welcome to Java* example (see the lecture slides).
Your task is to modify it: When it runs, it should greet you:

*Nice to see you, 'your name'!*
For example, *Nice to see you, Mr. Hassan!*

# Problem Two

Analyze the *Bicycle* and *BicycleDemo* classes of the **Implementation of a Bicycle** example (see the lecture slides). Your task is to propose the *Car* and *CarDemo* classes. Fields of the Car class are:

- horsePower (hp),
- speed (km/h),
- fuelConsumption (km/liter),
- gasTank (liter),
- travelledTime (h).

Methods of the Car class are:
constructor,
speedUp,
applyBrakes, t
ravelledTimeUp, and
printStateTank.

The last method should calculate the state of the *gasTank* taking into account *speed, fuelConsumption, travelledTime* and initial value of the *gasTank*. Consider that a car is travelling at a constant speed. You should create three *Car* objects in the *CarDemo* class. Please Specify different values for *speed* and *travelling time*. Your program should print the **states of tanks** of each car at the end of travelling time.

# END