



Overview of Object Oriented Programming and Java

OOP Lecture 1

Doyel Pal



Programs



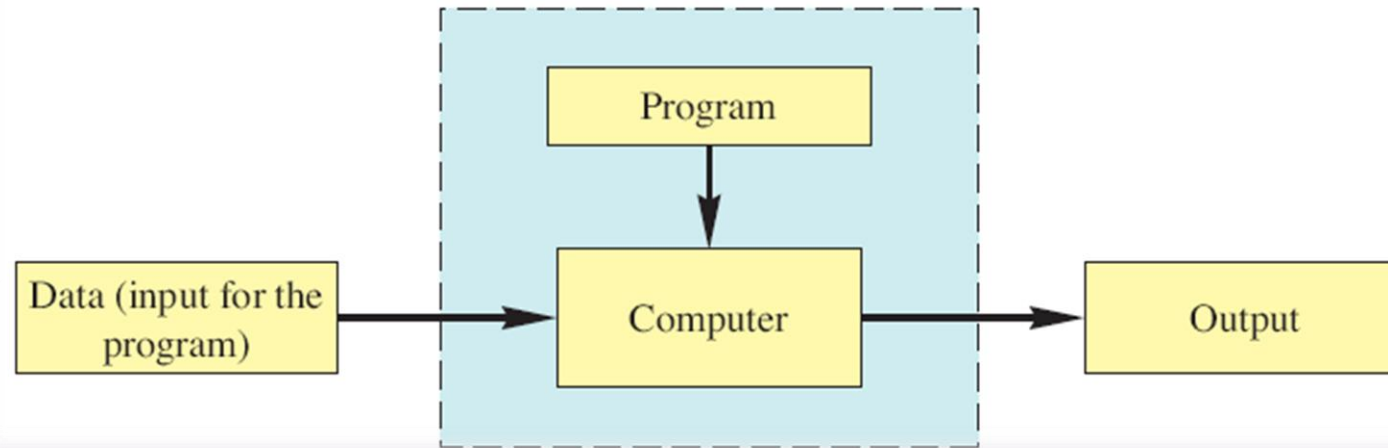
- A *program* is a set of instructions for a computer to follow.
- We use programs almost daily (email, word processors, video games, bank ATMs, etc.).
- Following the instructions is called *running* or *executing* the program.



Input and Output

- Normally, a computer receives two kinds of input:
 - The program
 - The *data* needed by the program.
- The output is the result(s) produced by following the instructions in the program.

Running a Program



- Sometimes the computer and the program are considered to be one unit.
 - Programmers typically find this view to be more convenient.



Programming Languages

- *High-level languages* are relatively easy to use
 - Java, C#, C++, Visual Basic, Python, Ruby.
- Unfortunately, computer hardware does not understand high-level languages.
 - Therefore, a high-level language program must be translated into a *low-level language*.



Compilers

- A *compiler* translates a program from a high-level language to a low-level language the computer can run.
- You *compile* a program by running the compiler on the high-level-language version of the program called the ***source program***.
- Compilers produce *machine-* or *assembly-language* programs called ***object programs***.
- Most high-level languages need a different compiler for each type of computer and for each operating system.
- Most compilers are very large programs that are expensive to produce.



Java Byte-Code

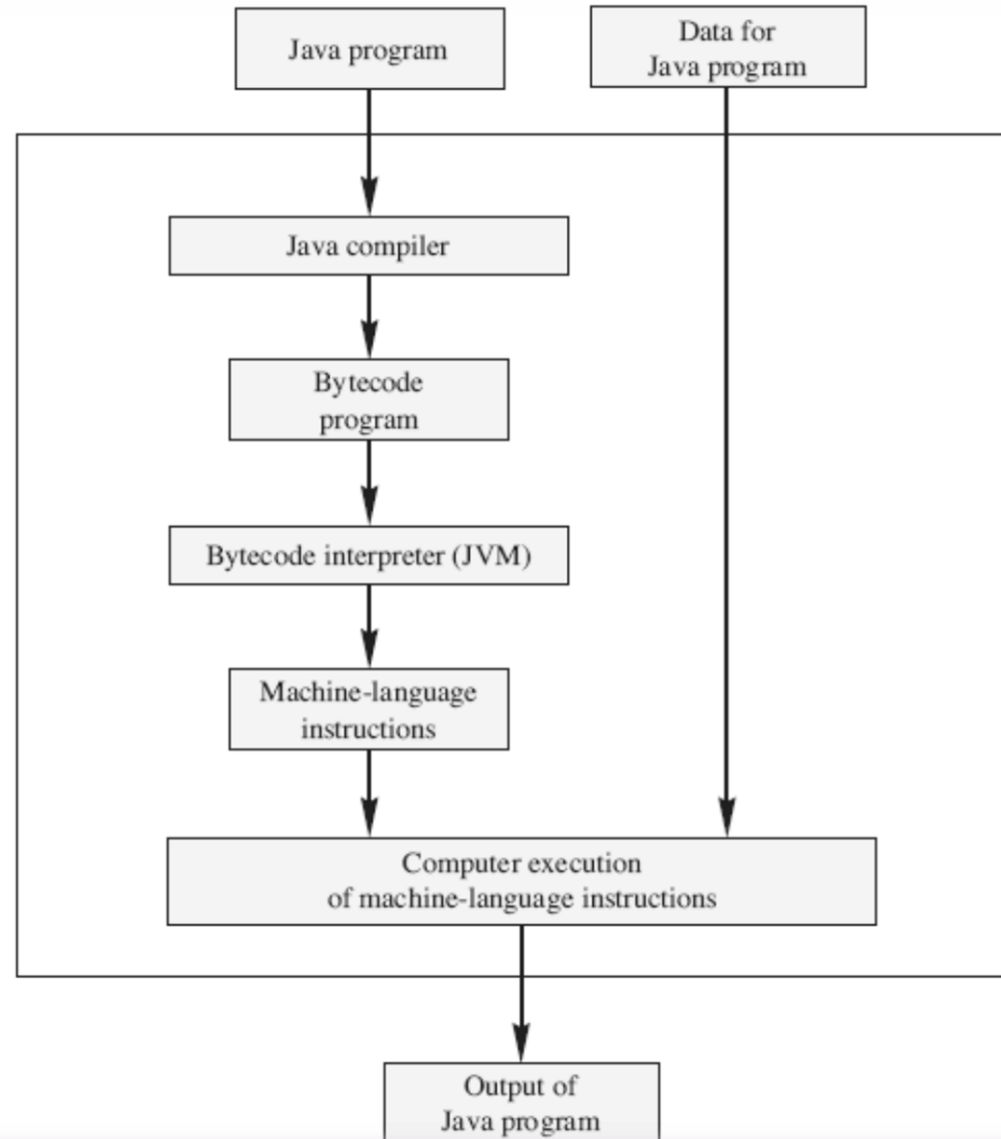
- The Java compiler **does not translate** a Java program into **assembly language or machine language** for a particular computer.
- Instead, it translates a Java program into **byte-code**.
 - Byte-code is the machine language for a hypothetical computer (or *interpreter*) called the **Java Virtual Machine**.



Java Byte-Code

- A **byte-code** program is easy to translate into machine language for any particular computer.
- A program called an **interpreter** translates each byte-code instruction, executing the resulting machine-language instructions on the particular computer before translating the next byte-code instruction.
- Most Java programs today are executed using a Just-In-Time or **JIT compiler** in which byte-code is compiled as needed and stored for later reuse without needing to be re-compiled.

Compiling and Running a Program



Compiling, Interpreting, Running

- Use the compiler to translate the Java program into byte-code (done using the **javac** command).
- Use the Java virtual machine for your computer to translate each byte-code instruction into machine language and to run the resulting machine-language instructions (done using the *java* command).



Portability

- After compiling a Java program into byte-code, that byte-code can be used on any computer with a byte-code interpreter and without a need to recompile.
- Byte-code can be sent over the Internet and used anywhere in the world.
- This makes Java suitable for Internet applications.



Class Loader

- A Java program typically consists of several pieces called **classes**.
- Each class may have a separate author and each is compiled (translated into byte-code) separately.
- A *class loader* (called a *linker* in other programming languages) automatically connects the classes together.

First Java Program

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

1. Create the program by typing it into a text editor and saving it to a file named, say, **"MyProgram.java"**.
2. Compile it by typing **"javac MyProgram.java"** in the terminal window.
3. Run (or execute) it by typing **"java MyProgram"** in the terminal window.



Some Terminology

- The item(s) inside parentheses are called **argument(s)** and provide the information needed by methods.
- A **variable** is something that can store data.
- An instruction to the computer is called a **statement**; it ends with a semicolon.
- The grammar rules for a programming language are called the **syntax** of the language.

Printing to the Screen

```
System.out.println ("Whatever you want to print");
```

- **System.out** is an object for sending output to the screen.
- **println** is a method to print whatever is in parentheses to the screen.



Compiling and Running

- Use an *IDE* (integrated development environment) which combines a text editor with commands for compiling and running Java programs.
- When a Java program is compiled, the byte-code version of the program has the same name, but the ending is changed from **.java** to **.class**.



Compiling and Running

- A Java program can involve any number of classes.
- The class to run will contain the words

```
public static void main(String[] args)
```

Sample Java Program


```
import java.util.Scanner;

public class FirstJavaProgram {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int a = input.nextInt();
        System.out.println("User given input is: "+a);
        String day = "This is my first Java Program";
        System.out.println(day);
    }
}
```



Programming Basics: Outline

- Object-Oriented Programming
 - Algorithms
 - Testing and Debugging
 - Software Reuse
- 



Programming



- Programming is a creative process.
- Programming can be learned by discovering the techniques used by experienced programmers.
- These techniques are applicable to almost every programming language, including Java.



Object-Oriented Programming



- Our world consists of *objects* (people, trees, cars, cities, airline reservations, etc.).
- Objects can perform *actions* which affect themselves and other objects in the world.
- Object-oriented programming (OOP) treats a program as a collection of objects that interact by means of actions.



OOP Terminology

- Objects, appropriately, are called *objects*.
- Actions are called *methods*.
- Objects of the same kind have the same *type* and belong to the same *class*.
 - Objects within a class have a common set of methods and the same kinds of data but each object can have it's own data values.


OOP Design Principles



- OOP adheres to three primary design principles:
 - Encapsulation
 - Polymorphism
 - Inheritance



Introduction to Encapsulation

- 
- The data and methods associated with any particular class are encapsulated (“put together in a capsule”), but only part of the contents is made accessible.
 - Encapsulation provides a means of using the class, but it omits the details of how the class works.
 - Encapsulation often is called *information hiding*.

Encapsulation Example

```
public class Student{  
    private String name;  
  
    public String getName(){  
        return name;  
    }  
    public void setName(String name)  
    {  
        this.name=name  
    }  
}
```

```
class Test{  
    public static void main(String[] args)  
    {  
        Student s=new Student();  
        s.setname("Peter");  
        System.out.println(s.getName());  
    }  
}
```

No outside class can access private data member (variable) of other class.
Outside class can access private data fields via public methods.

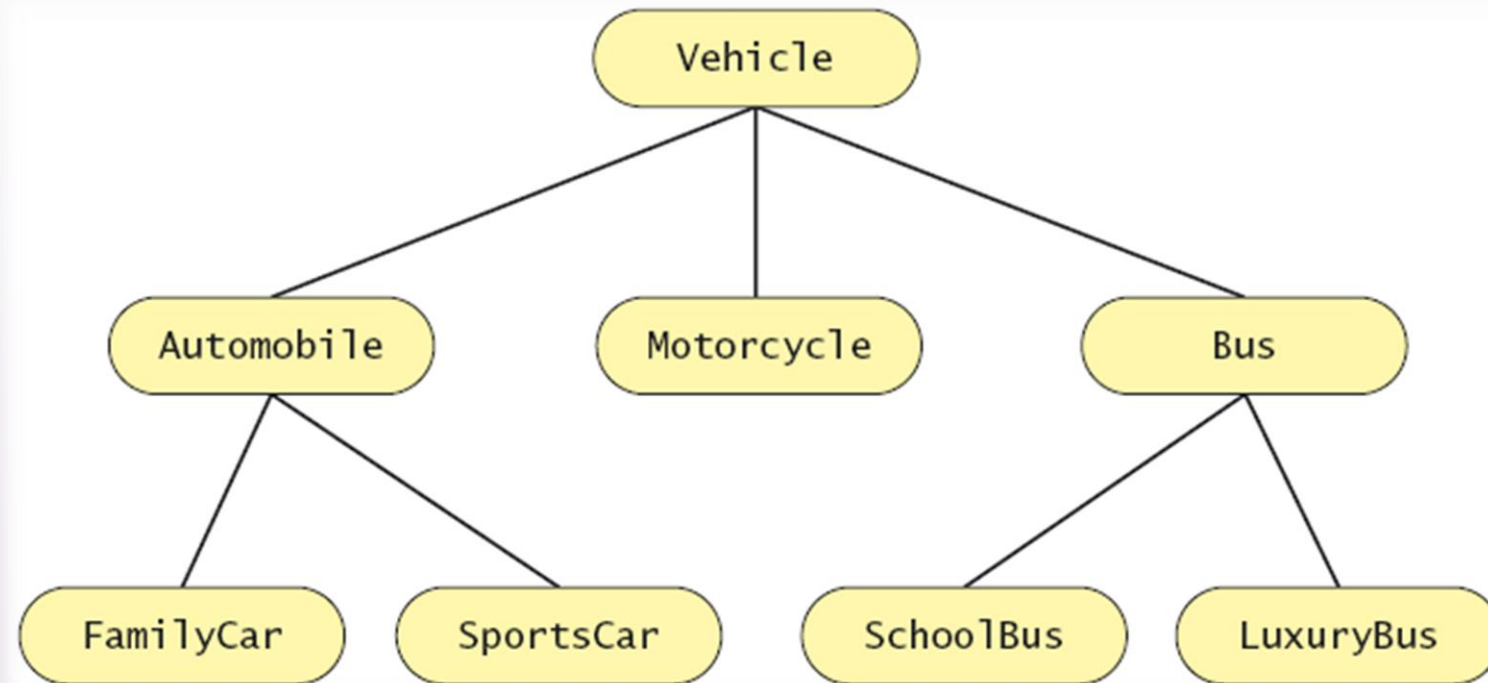
Data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes. That's why encapsulation is known as **data hiding**.



Introduction to Polymorphism

- From the Greek meaning “many forms”
- The same program instruction adapts to mean different things in different contexts.
 - A method name, used as an instruction, produces results that depend on the class of the object that used the method.
 - Everyday analogy: “take time to recreate” causes different people to do different activities.

Introduction to Inheritance





Introduction to Inheritance

- Classes can be organized using *inheritance*.
- A class at lower levels inherits all the characteristics of classes above it in the hierarchy.
- At each level, classifications become more specialized by adding other characteristics.
- Higher classes are more inclusive; lower classes are less inclusive.



Inheritance in Java

- Used to organize classes
- “Inherited” characteristics do not need to be repeated.
- New characteristics are added.



Algorithms



- By designing methods, programmers provide actions for objects to perform.
- An *algorithm* describes a means of performing an action.
- Once an algorithm is defined, expressing it in Java (or in another programming language) usually is easy.



Algorithms



- An algorithm is a set of instructions for solving a problem.
- An algorithm must be expressed completely and precisely.
- Algorithms usually are expressed in English or in *pseudocode*.



Reusable Components

- Most programs are created by combining components that exist already.
- Reusing components saves time and money.
- Reused components are likely to be better developed, and more reliable.
- New components should be designed to be reusable by other applications.



Testing and Debugging

- Eliminate errors by avoiding them in the first place.
 - Carefully design classes, algorithms and methods.
 - Carefully code everything into Java.
- Test your program with appropriate test cases (some where the answer is known), discover and fix any errors, then retest.



Errors

- An error in a program is called a *bug*.
- Eliminating errors is called *debugging*.
- Three kinds of errors
 - Syntax errors
 - Runtime errors
 - Logic errors



Syntax Errors

- Grammatical mistakes in a program
 - The grammatical rules for writing a program are very strict
- The compiler catches syntax errors and prints an error message.
- Example: using a period where a program expects a comma



Runtime Errors

- Errors that are detected when your program is running, but not during compilation
- When the computer detects an error, it terminates the program and prints an error message.
- Example: attempting to divide by 0



Logic Errors

- Errors that are not detected during compilation or while running, but which cause the program to produce incorrect results
- Example: an attempt to calculate a Fahrenheit temperature from a Celsius temperature by multiplying by $9/5$ and adding 23 instead of 32



Software Reuse

- Programs not usually created entirely from scratch
- Most contain components which already exist
- Reusable classes are used
 - Design class objects which are general
 - Java provides many classes



Summary

- You have completed an overview of the Java programming language.
 - You have been introduced to program design and object-oriented programming.
- 