Prerequisite Topics for Advanced Java Course

for

Master In Information Technology
InfoLink University College

by

Elias Jarso

Introduction to OOP AND

Contents

- Programming Paradigms
- Introducing Object Oriented Programming
- Basic Features of OOP
- Introduction to Java
- Characteristics of Java
- Java Virtual Machine
- Phases of Creating and Executing Java Code

Programming Paradigms

Programming Paradigm can be expressed in many ways:

- It is a model of programming based on distinct concepts that shapes the way programmers design, organize and write programs
- It is an approach or philosophy of programming that programmers follow while developing an application.
- A way of conceptualizing what it means to perform computation and how tasks to be carried out on the computer should be structured and organized.
- There are so many programming paradigms: Unstructured Programming Paradigm, Procedural Programming Paradigm and Object Oriented Programming Paradigm. These are the well known paradigms.

Unstructured Programming

- It is where the whole program is written within a single algorithm.

Procedural Programming

- In this approach a given problem is viewed as sequence of things to be done such as reading, calculating, printing, registering etc.
- A number of functions are then written to accomplish these tasks. Hence, a program is broken down to smaller units each performing different tasks, this units are called Procedures.
- Procedure is nothing but series of computational steps to be carried out. It can also be referred to as routines, subroutines, methods, or functions.
- Often thought as a synonym for **imperative programming** (Specifying the steps the program must take to reach the desired state)
- In PoP the primary focus is on functions. The following figure explains the concept.

Procedural Programming

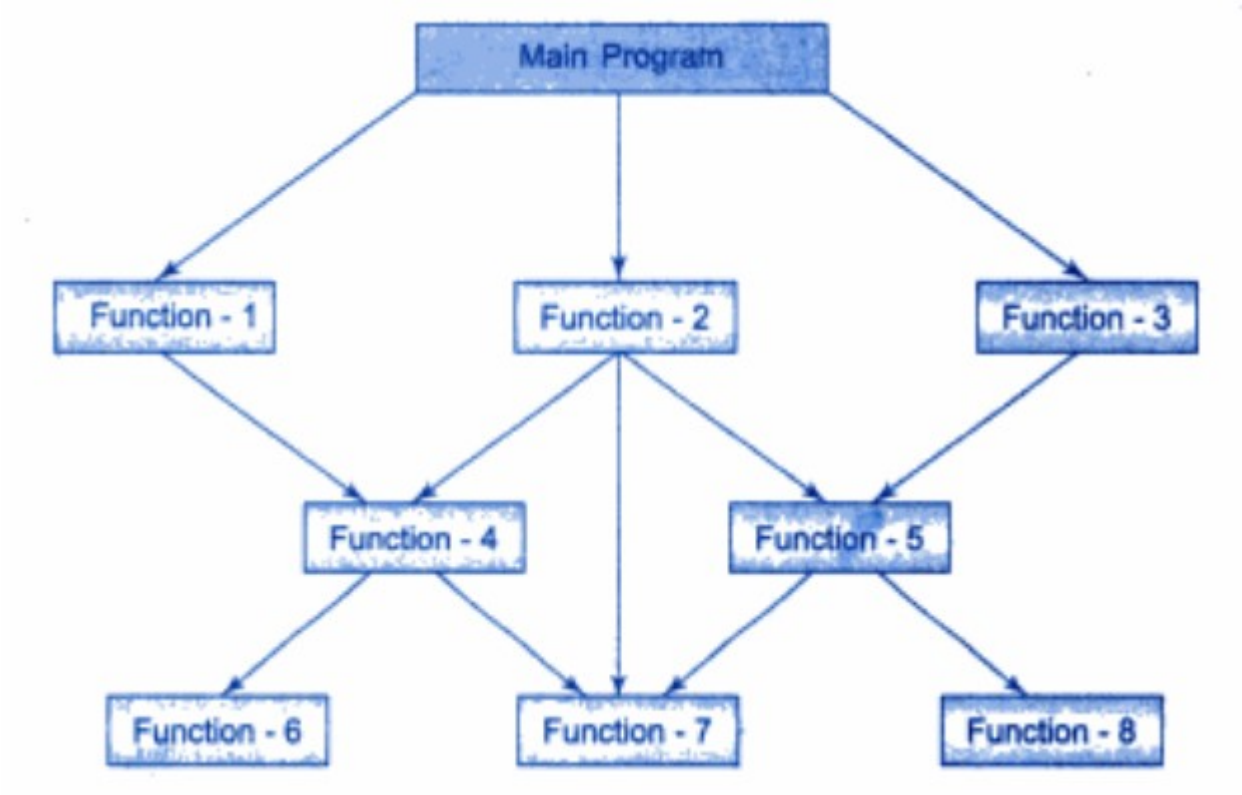


Fig: Typical Structure of Procedure Oriented Programming

Procedural Programming

- In multi-function program, many important data items are placed as global so that they can be accessed by all the functions which need them.

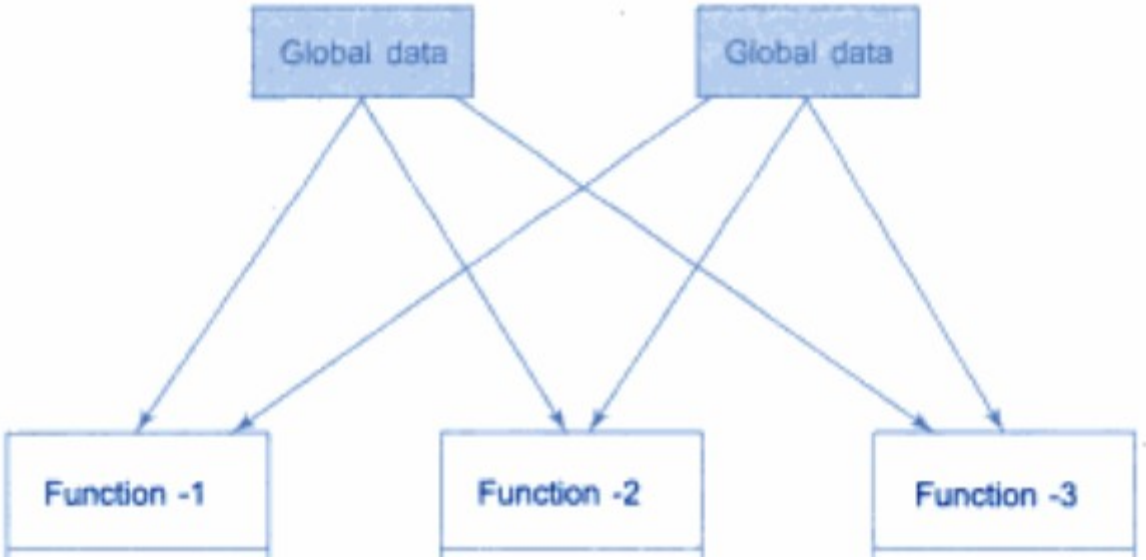


Fig: Relationship between data and function in PoP

Procedural Programming

Problems with PoP:

- Global data are more vulnerable to inadvertent (or unintentional) change by a function.
- Since the data are global and can be accessed by any function, irrelevant data may simply be used by inappropriate function.
- Besides, if we want to modify a global data, since there is a possibility of error occurring in those functions using the data, we need to revise all the function using that data.
  - **In large program, since there are lots of global data and many functions, there is a complex web between the data and the functions. Hence, figuring out those functions using the data is a tough time consuming work in addition to the possibility of allowing bugs to be created.**
- Another serious drawback with the PoP is that it doesn't model real world problems very well. One reason for this is functions and data are independent.

Object Oriented Programming

- Object-oriented programming (OOP) is a programming paradigm that uses "objects" – which is data structures encapsulating data fields and functions together with their interactions – to design applications and computer programs.
- OOP treats data as critical element in program development and doesn't allow it to flow freely (globally) around the system.
- OOP ties data more closely to the functions which operate on them.
- OOP allows decomposition of a problem into a number of entities called *objects* and then builds data and functions around these objects.
  - For example a given object of a **Student** type may posses data such as **id**, **name**, **sex**, **age**, **gpa** AND functions such as **register(), addCourse(), generateGradeReport()** etc. These data and function will then be put under the same unit.
- The data of an object can be accessed only by the functions associated with that object.

Object Oriented Programming

- Unlike PoP, OOP models real world problems very well. Because in real world thinking humans portray (or define) almost everything as Object (Student, Room, Car, Tree, etc.)
- OOP is good at modelling real-world objects in software
- Why design applications in this way?
  - We naturally *class*ify objects into different *types*.
  - By attempting to do this with software aim to make it more **maintainable**, **understandable** and **easier to reuse.**
- Many modern programming languages now support OOP.

  Eg. Java, C++, C#

Procedural vs OO Programming

- The unit in procedural programming is function, and unit in object-oriented programming is class.
- Procedural programming concentrates on creating functions while object-oriented programming starts from isolating the classes, and then look for the data and methods inside them.
- Procedural programming separates the data of the program from the operations (functions) that manipulate the data, while object-oriented programming focus on both of them and encapsulate them.

Basic Concepts (or Features) of OOP

- Concepts used extensively in Object Oriented Programming are:
  - Classes
  - Objects
  - Encapsulation
  - Data Hiding
  - Abstraction
  - Inheritance
  - Polymorphism

A. Classes and Objects

- Classes
  - **“Class” is a blueprint(or a design).**
  - It defines the variables and methods/functions the objects support. In other words,
    - ✓ It is a blueprint or a design based on which objects are made.
    - ✓ It is a design which defines what objects should contain or look like.
    - ✓ It is also a data type based on which objects are created.
- Object
  - **“Object” is an instance of a class. Each object has a class which defines its data and behavior.**
  - It is also a basic runtime entity of an Object Oriented System.
  - It is a variable created using a Class as a type.

More on OOP

- Computer scientists have introduced the notion of **objects** and **object-oriented programming** to help manage the growing complexity of modern computers.
- An **object** is **anything that can be represented by data** in a computer's memory and manipulated by a computer program. It is considered to be a partitioned area of computer memory that stores data.
- To a computer, an object is simply something that can be represented by data in the computer's memory and manipulated by computer programs.
- An **object** can be something in the **physical world** or even just an **abstract idea**.
  - An airplane, for example, is a physical object that can be manipulated by a computer.
  - A bank transaction is an example of an object that is not physical.

More on Classes and Objects

- The data that represent the object are organized into a set of ***properties***. (For Example: id, name, sex, department, section, gpa, classyear etc are properties of a student object)

ID: CS/346/07
Name: Elias
Sex: Male
Department: Computer Science
Section: Three
GPA: 3.8
ClassYear: Second Year

- The values stored in an object's properties at any one time form the ***state*** of an object. (For Example: from the above table the state of the current object are "CS/346/07", "Elias", "Male", "Computer Science", "Three", "3.8", "Second Year")

More on Classes and Objects

- Computer programs implement algorithms that manipulate the data.
- In object-oriented programming, the programs that manipulate the properties of an object are the object's **methods**.
- We can think of class as a collection of properties and the methods that are used to manipulate those properties.

Properties

Methods

```
class Bicycle {

     int cadence = 0;
     int speed = 0;
     int gear = 1;

     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;
     }
```

More on Classes and Objects

- Each copy of an object from a particular class is called an **instance** of the class. (E.g. “Dawit” can be one instance of Student class while “Helen” is another instance)
- The act of creating a new instance of an object is called **instantiation.**
- Two different instances of the same class will have the same properties, but different values stored in those properties.
  - For example every student object possess same properties such as name, sex, gpa but every object have its own value “Jemal, Male, 3.2” or “Helen, Female, 3.4”
- ….

More on Classes and Objects

Object

Property

The same terminology is used in most object-oriented programming languages.

Instantiation

Method

Class

Instance

State

More on Classes and Objects

- Example of a class

```
class Student {
        String name;
        char sex;
        float gpa;

        void setName(String n){
           name = n;
        }
        void setSex(char s){
           sex = s;
        }
        void setGPA(float g){
           gpa = g;
        }
        void displayInfo(){
           System.out.println("Name:  " + name );
           System.out.println("Sex:  " + sex);
           System.out.println("GPA:  " + gpa);
        }

}
```

- ...

More on Classes and Objects

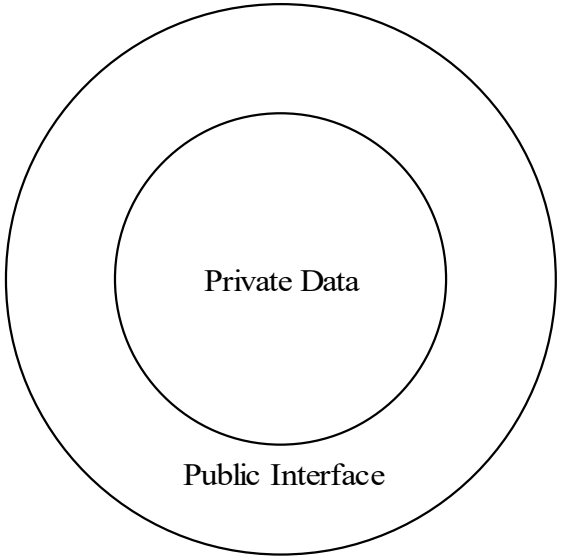
- **A class can have three kinds of members:**
  - ***fields***: data variables which determine the status of the class or an object
  - ***methods***: executable code of the class built from statements. It allows us to manipulate/change the status of an object or access the value of the data member
  - ***nested classes and nested interfaces***
- We can include any one or all of these as a member of a class as we like

```
class Student {
        - fields/variables as member
        - methods/functions as member
        - Another class and/or interfaces as member
}
```

B. Encapsulation

- Wrapping up of data and functions into a single unit(called class) is known as Encapsulation.
  - Data is not accessible to the outside world and only those function which are wrapped in the class can access it.
  - If any outside program wants to use the data, it can get the data only through those wrapped functions. Therefore, the functions wrapped in the class provide an interface(or a way) between the data and the program.
    - **This insulation of the data from direct access by outside programs is called Data Hiding or Information Hiding.**
- The data (state) of an object is private – it cannot be accessed directly. The data (state) can only be changed or accessed through its behaviour, otherwise known as its public interface or contract (or simply the inside functions). This is called encapsulation.

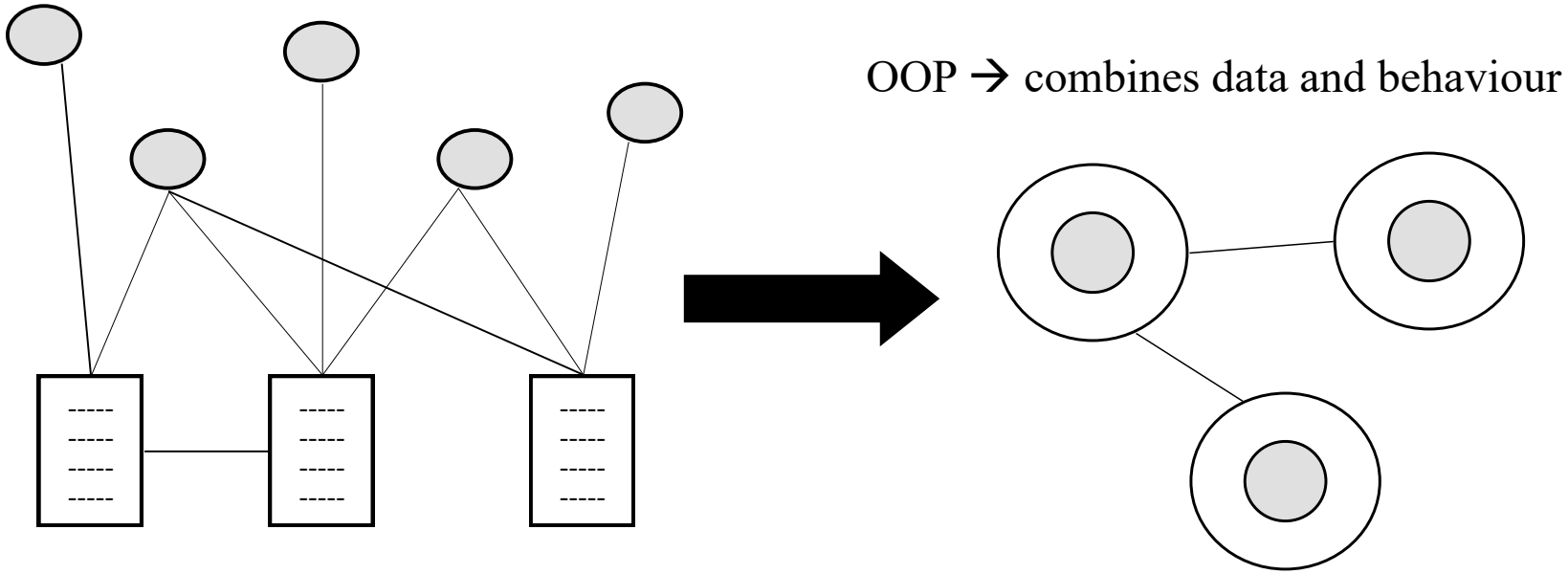
B. Encapsulation



This shows that an object has a private state and public behavior. Private states can only be changed by invoking some behavior.

- The main benefit of encapsulation is
  - **Internal state and processes can be changed independently of the public interface**
  - **Limits the amount of large-scale changes required to a system**

B. Encapsulation



"Conventional programming"
functions operating on independent data

C. Abstraction

- It refers to the act of representing essential features without including the background details or explanations.
- Classes are abstract data types.
- Abstract Data Types:
  - Identifying abstract types is part of the modelling/design process
    - The types that are useful to model may vary according to the individual application
    - For example a payroll system might need to know about Departments, Employees, Managers, Salaries, etc
    - An E-Commerce application may need to know about Users, Shopping Carts, Products, etc
  - Object-oriented languages provide a way to define abstract data types, and then create *objects* from them
    - It's a template (or 'cookie cutter') from which we can create new objects
    - For example, a Car class might have attributes of speed, colour, and behaviours of accelerate, brake, etc
    - An individual Car *object* will have the same behaviours but its own values assigned to the attributes (e.g. 30mph, Red, etc)

D. Inheritance

- Inheritance is the ability to define a new class in terms of an existing class
  - The existing class is the *parent*, *base* or *superclass*
  - The new class is the *child*, *derived* or *subclass*
- The child class inherits all of the attributes and behaviour of its parent class
  - It can then add new attributes or behaviour
  - Or even alter(change) the implementation of existing behaviour
- Inheritance is therefore a method of code reuse

E. Aggregation

- Aggregation is the ability to create new classes out of existing classes
  - Treating them as building blocks or components (i.e. Object of one class type can be a member of another class).
- Aggregation is another way of code reuse
- Two forms of aggregation
  - **Whole-Part relationships**
    - Car is made of Engine, Chassis, Wheels
  - **Containment relationships**
    - A Shopping Cart contains several Products
    - A List contains several Items

F. Polymorphism

- Polymorphism means ‘many forms’
- In brief though, polymorphism allows two different classes to respond to the same message in different ways
  - E.g. both a Plane and a Car could respond to a ‘turnLeft’ message, however the means of responding to that message (turning wheels, or banking wings) is very different for each.
  - Or Simple example could be both Circle and Rectangle could have a function ‘findArea’, but the way they do it is different. Rectangle finds area by multiplying length by width while circle multiplies pi by square of its radius.
- Allows objects to be treated as if they’re identical

Summary

- In OO programming we
  - Define classes
  - Create objects from them
  - Combine those objects together to create an application
- Benefits of OO programming
  - Easier to understand (closer to how we view the world)
  - Easier to maintain (localised changes)
  - Modular (classes and objects)
  - Good level of code reuse (aggregation and inheritance)

Summary

- Understanding OOP is fundamental to writing good Java applications
  - Improves design of your code
  - Improves understanding of the Java APIs
- There are several concepts underlying OOP:
  - Abstract Types (Classes)
  - Objects
  - Encapsulation (or Information Hiding)
  - Aggregation
  - Inheritance
  - Polymorphism

Introduction to

History of Java

- Java is a general purpose, object oriented programming language developed by Sun Microsystems in 1991 as part of a **research work** to develop software for consumer electronics. (TV Sets, Toaster, Digital Cameras, DVD layer, Video Game Consoles, Phones, etc.)
- The main aim was to create a **simple**, **portable** and **reliable** language.(Java was used to add dynamic content to web 93')
- Modelled after C++, much of syntax and object oriented structure is borrowed from C++.
- Originally named **Oak** by James Gosling (one of the inventors of Java) – after oak tree outside their window.
- Later on renamed as **Java**
- Owned by Oracle since 2010.

History of Java

- Java enables users to develop and deploy applications on the Internet for Servers, Desktop Computers, and Small Hand-Held devices.
  - **Simply put, Java can be used to develop**
    - **Desktop Applications**
    - **Java Applets**
    - **Web Applications**
    - **Applications for Hand-Held Devices such as Palm and Cell Phones**

Characteristics of Java

- Simple
- Object-Oriented
- Distributed
- Compiled and Interpreted
- Robust
- Secure
- Architecture-Neutral
- Portable
- Performance
- Multithreaded
- Dynamic

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java is partially modeled on C++, but greatly simplified and improved. Some people refer to Java as "C++--" because it is like C++ but with more functionality and fewer negative aspects.

Java omits many rarely used, poorly understood, confusing features of C++ that, inexperience, bring more grief than benefit.

- ✓ No header files
- ✓ No pointers
- ✓ No structure or Union
- ✓ No operator overloading
- ✓ No multiple inheritance
- ✓ No global variables

Because Java is simple, it is easy to read & write code

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java is inherently object-oriented.

One of the central issues in software development is how to reuse code. Object-oriented programming provides great flexibility, modularity, clarity, and reusability through encapsulation, inheritance, and polymorphism.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Distributed computing involves **several computers working together on a network.** Java is designed to make distributed computing easy. Since networking capability is inherently integrated into Java, writing network programs is like sending and receiving data to and from a file.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java compiler translates Java code to Bytecode instruction. (not a native machine code, it is a Java Virtual Machine code)

The byte code is machine-independent and can run on any machine that has a Java Interpreter, which is part of the Java Virtual Machine (JVM).

Java Interpreter generates Machine Code that can be directly executed by host machine. So, you need an interpreter to run Java programs.

Unlike machine language Java Bytecode is exactly the same on every platform.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust(Safe)
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java implements robust exception handling. Java compilers can detect many problems that would first show up at execution time in other languages.

Java is defined as a **garbage-collected language**, because it helps programmers virtually from all memory management problems.

Java has **Strong Typing.**

Java has eliminated certain types of error-prone programming constructs found in other languages.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust(Safe)
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java implements several security mechanisms to protect your system **against harm caused by stray programs**.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust(Safe)
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Write once, run anywhere!!

A key goal of Java is to be able to write programs that will run on a great variety of computer systems and computer-controlled devices.

With a Java Virtual Machine (JVM), you can write one program that will run on any platform. So it is platform independent.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust(Safe)
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust(Safe)
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java's performance Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.

Java bytecode rivals C++ code in speed.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust(Safe)
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Benefits of multithreading are **better interactive, responsiveness and real-time behavior.**

A single Java program can have many different threads running independently and continuously.

Multithread programming is smoothly integrated in Java, whereas in other languages you have to call procedures specific to the operating system to enable multithreading.

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust(Safe)
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java was designed to adapt to an evolving environment. New code can be loaded on the fly without recompilation. There is no need for developers to create, and for users to install, major new software versions. New features can be incorporated transparently as needed.

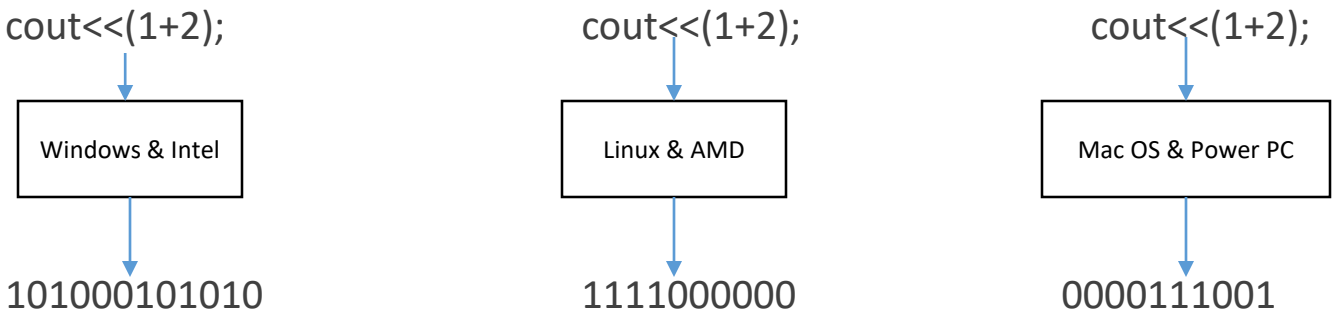
The Java Virtual Machine **(JVM)**

- Java is not only a programming language but also a **Platform.**
  - A platform is the hardware or software environment in which a program runs.
  - Or a plat form can be said to be operating system & underlying hardware
  - E.g. of common platforms are:
    - Windows & Intel (Commonly called **Wintel**)
    - Linux & AMD
    - Mac OSx & Power PC(IBM)
- Java platform has two components:
  1. **The Java Virtual Machine (JVM)** – which is software based machine
  2. **The Java Application Program Interface (API)** – which provides software env't
     - **API** is a large collection of **ready-made software components** that provides many useful capabilities. It is grouped into libraries of related classes and interfaces; this libraries are called packages.

  - **The Java platform is different because it is software-only platform that runs on top of other hardware based platforms.**

The Java Virtual Machine **(JVM)**

- Consider compilation of other language (C++)
  - E.g. the result of compilation on different platform

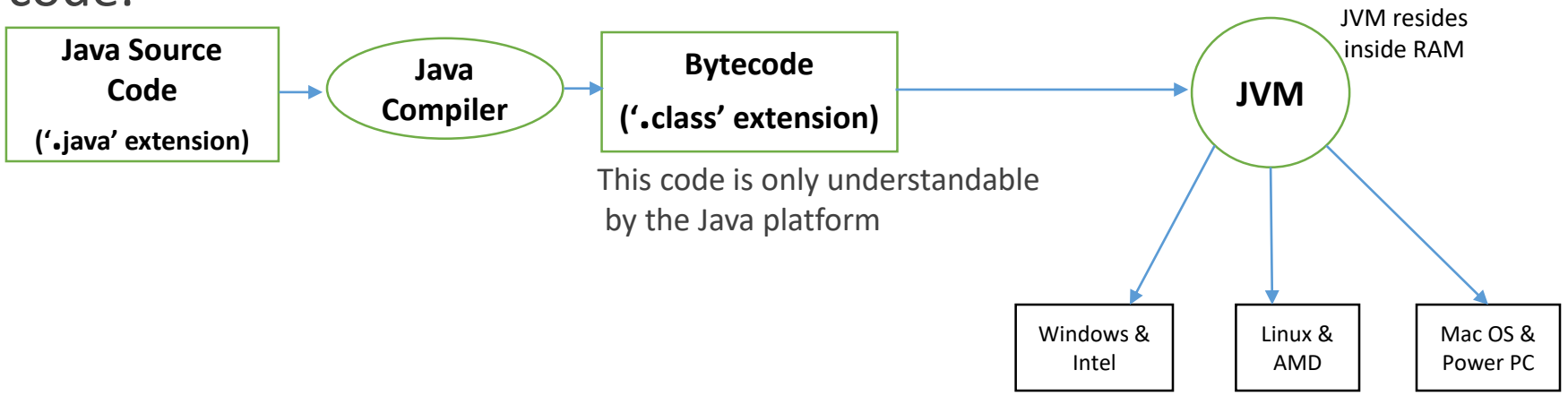


  - Notice the compilation generates different result for different platform
  - With change in platform, the machine language changes.
  - **As developer we want our software to work on all platforms available. So, we need to have different compilers and re-compile our code.**
- **How about Java?**

The Java Virtual Machine (JVM)

- All compilers for many other languages convert Source Code to Machine Code. But Java compiler converts Source Code to intermediate code called Bytecode.

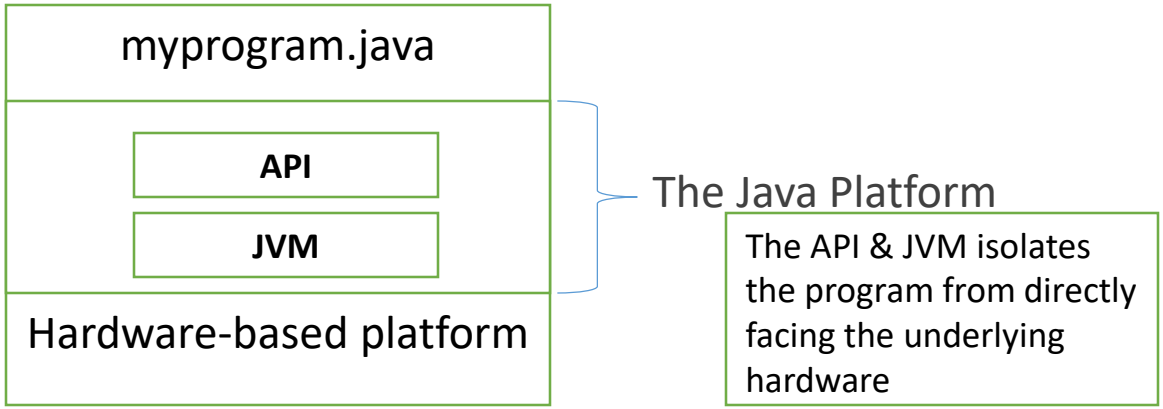
- Once JVM is given bytecode, it identifies which platform it is working on and converts the bytecode into native machine code.



  - JVM uses Interpreter and other tools to generate respective machine code for each platform.

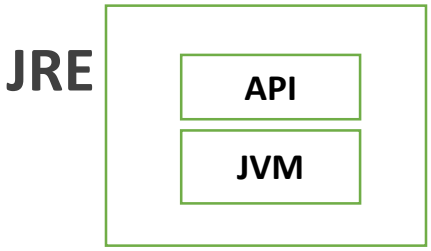
The Java Virtual Machine **(JVM)**

- The Java code once compiled not only run on all PC platforms but also mobiles & other electronic gadgets supporting Java.
- **Furthermore, Java is Free to use.**



- **JRE** (Java Runtime Environment) is part of the Java Development Kit (JDK).
  - **JDK** is a set of programming tools for developing Java application. It is what you download and install to when working with Java.
  - **JRE** provides minimum requirement for executing a Java application. JRE consists of: JVM, Core classes and other supporting files.

The Java Virtual Machine (JVM)



- **Components of JVM:**
  1. **Class Loader**
  2. **Bytecode Verifier**
  3. **Execution Engine**
  4. **Security Manager**
  5. **Garbage Collector**
- **JDK** consists of **JRE** plus **command-line development tools** such as compilers and debuggers that are necessary for developing Java applications
- **JRE consists of**: JVM, Core classes(API) and other supporting files.
- **Assignment:**
  - ↳ **Explain in detail roles and responsibilities of each JVM components.**

JDK Editions

- **Java Standard Edition (J2SE)**
  - J2SE can be used to develop client-side standalone applications or applets.
- **Java Enterprise Edition (J2EE)**
  - J2EE can be used to develop large-scale, distributed networking

JDK=JRE + commandline tools
(compiler, Javadoc, debugger)
JRE = API + JVM

JVM =
- Class Loader
- Bytecode Verifier
- Execution Engine
- Security Engine
- Garbage Collector

Phases of Creating & Executing Java

- There are five phases
  1. Edit
  2. Compile
  3. Load
  4. Verify
  5. Execute

Phases of Creating & Executing Java

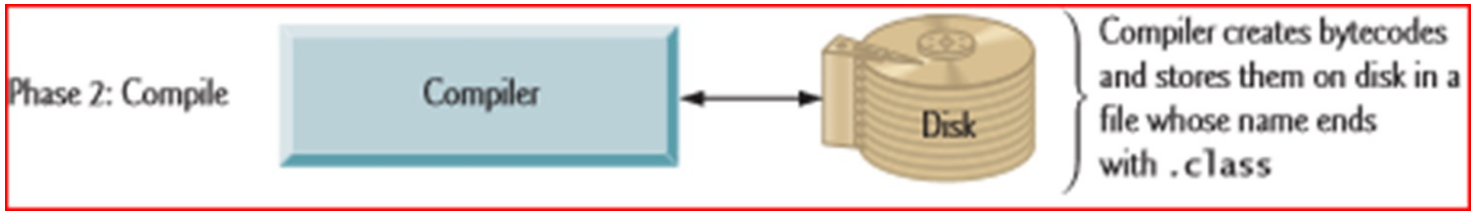
1. Editing

- Consists of editing a file with an **editor**.
- Using the editor, you type a Java program (typically referred to as **source code**), make any necessary corrections and save it on a secondary storage device, such as your hard drive.
- Java source code files are given a name ending with the .java **extension**, indicating that the file contains Java source code.
- The editor can be simple editors like **notepad** or advanced once called **IDE (Integrated Development Environment)**
  - Integrated development environments (IDEs) provide tools that support the software development process, such as **editors**, **debuggers** for locating errors.

Phases of Creating & Executing Java

2. Compiling

- Java compiler compiles Java Source Code into Bytecode
- In this phase we use the command **javac** (the Java compiler) to compile a program
- The Java compiler translates **Java source code** into **bytecodes** that represent the tasks to execute in the execution phase.
  - **The Java Virtual Machine (JVM),** which is a part of the JDK and the foundation of the Java platform, **executes byte codes.**

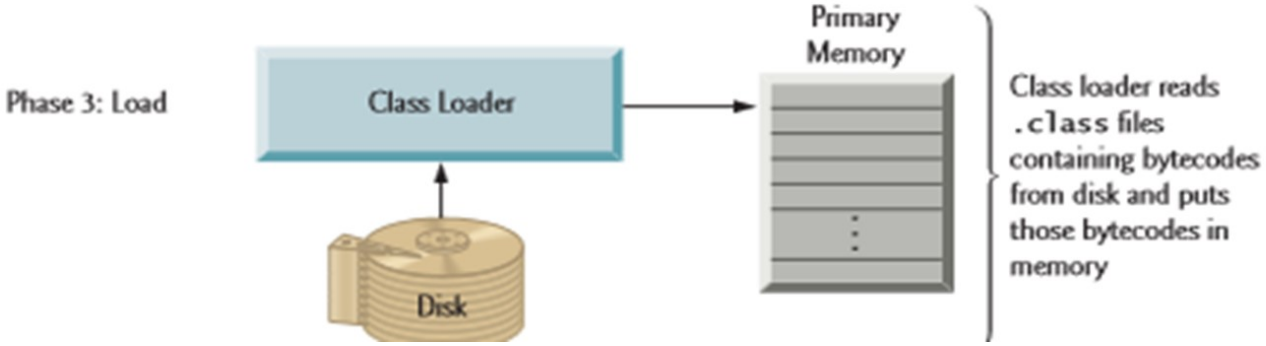


  - So, portability, platform independence or architecture neutrality is achieved here. HOW?

Phases of Creating & Executing Java

3. Loading

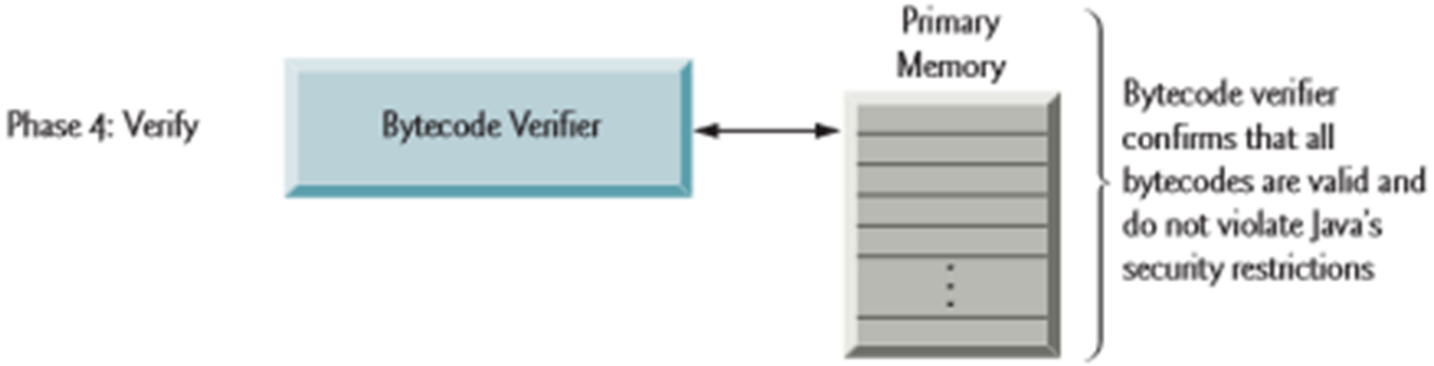
- This is simply loading the program into memory (main memory)
- the JVM places the program in memory to execute it—this is known as *loading*.
- The JVM's class loader takes the **.class** files containing the program's bytecodes and transfers them to primary memory. It also loads any of the .class files provided by Java that your program uses.
- The **.class** files can be loaded from a disk on your system or over a network (e.g., your local college or company network, or the Internet).



Phases of Creating & Executing Java

4. Verifying

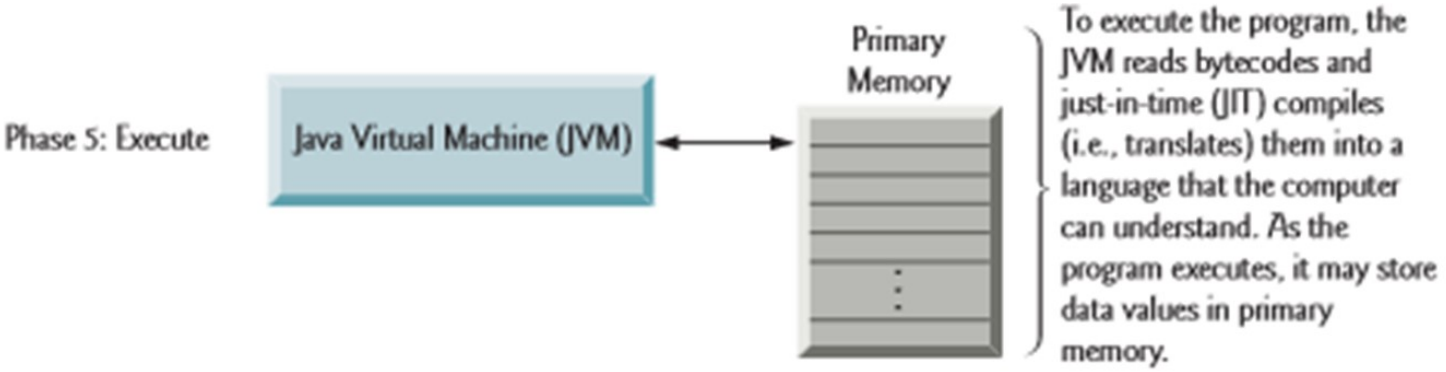
- As the classes are loaded into memory, the bytecode verifier examines their bytecodes to ensure that they're valid and do not violate Java's security restrictions.
- Java enforces strong security to make sure that Java programs arriving over the network do not damage your files or your system (as computer viruses and worms might).



Phases of Creating & Executing Java

5. Executing

- The JVM executes the program's bytecode using Interpreter and other tools
- In early Java versions, the JVM was **simply an interpreter** for Java bytecodes. Most Java programs would execute slowly, because the JVM would only interpret and execute one bytecode at a time.
- Today's JVMs typically execute bytecodes using a **combination of interpretation and so-called just-in-time (JIT) compilation**. ***What is JIT?***



Phases of Creating & Executing Java

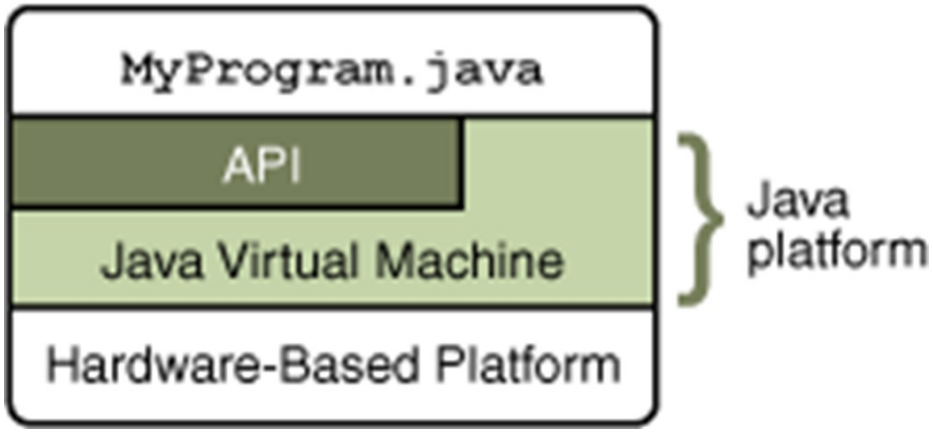
- **JIT -** just-in-time (JIT) compilation
  - The JVM executes the program's byte codes.
  - JVM typically uses a combination of interpretation and just-in-time (JIT) compilation.
  - **JVM analyzes the bytecodes as they are interpreted, searching for hot spots— meaning parts of the byte codes that execute frequently.**
  - **A just-in-time (JIT) compiler (the Java Hot Spot compiler) translates the bytecodes into the underlying computer's machine language.**
  - When the JVM encounters these compiled parts again, the faster machine-language code (which has been compiled) executes.
  - So, Java programs actually go through two compilation phases
    - **One in which source code is translated into byte codes (for portability across computer platforms)**
    - **A second in which, during execution, the byte codes are translated into machine language for the actual computer on which the program executes.**

Extra Points

- **Where do we find the JDK ?**
  - You can download the latest version of JDK from:
  - ✓ http://www.oracle.com/technetwork/java/javase/downloads/index.html
- **What are popular IDEs and Where to get them:**
  - Eclipse (www.eclipse.org)
  - NetBeans (www.netbeans.org)
  - JBuilder (www.codegear.com)
  - JCreator (www.jcreator.com)
  - BlueJ (www.blueJ.org)
  - jGRASP (www.jgrasp.org)
  - InteliJ

Review Assignment

1. Discuss and specify each one and the components of JVM, JRE and JDK with their functionality in detail.
2. What does the following figures represent? Explain it in detail.



Chapter 2
Basics of Java

Contents

Hello World Program

- **It is a simple Java code to show what Java code structure look like**

```
/**
 * The HelloWorld class implements an application that
 * simply prints "Hello World!" to standard output.
 */
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello World!");
  }
}
```

Hello World Program

- **Note:**

1. The line "**public class HelloWorld**" explanation:
   - **'class'** keyword is used to create any class in Java. In this particular example we created a class named "**HelloWorld**"
   - The class used an access-specifier called "**public**" which indicates that our class is accessible to other packages. (Simply to mean that our class can be used by other classes in other folders). If we removed the "public" specifier it means that our class can only be used by other classes in the same folder.

Hello World Program

- **Note:**

  2. The line "**public static void main(String[] args)**"
     - This is the begging line for definition of the method "**main**"
     - The "main" method is the starting point of for execution of any Java program. It is where the JVM begins execution of the program.
     - The main method takes String array as parameter "String[] args". This is used for array arguments passed by command line.
     - **The main method should always follow the above signature.**
       - ✓ The modifiers public and static can be written in either order (public static or static public), but the convention is to use public static as shown above.
       - ✓ You can name the argument anything you want, but most programmers choose "args" or "argv".
     - **NB: Java Applets do contain the main method.**

Hello World Program

- **Note:**

  3. The line **System.out.println("Hello World!")**
     - does same job as **cout** in C++
     - Since Java is a truly OO language, every method must be part of (or member of) an object. The "**println**" method is a member of an "**out**" object. The "**out**" object itself is a static member of class "**System**". So the proper way to use the function is: **System.out.println()**
     - The method "println" prints a given text to the screen.
     - The method "println" also appends new line to the end of the text. But the "print" method does not. 'System.out.print()'

Another Example Program

- **Another example which prints square root of a number**

```
import java.lang.Math; //optional

class Demo{

    public static void main(String[] args) {

        double x = 5;

        double y;

        y = Math.sqrt(x);

        System.out.println("Square root is: " + y);
    }

}
```

- The "**sqrt**" method is a static member of the "**Math**" class.
- The line "import java.lang.Math" includes the Math class for our use. But this line is optional since Math class is in "java.lang" package.

Java Comments

- **Comments**: are notes written in the source code by the programmer to make the code easier to understand.
  - Comments are ignored by the compiler but are useful to other programmers.
  - Comments are not executed when your program runs.
  - Most Java editors show your comments with a special color.
- **Java supports three kinds of comments:**
  - **Single-Line Comment - //**
  - **Multiple-Line Comment - /* ……. */**
  - **Documentation Comment (Javadoc Comment) - /**……. */**
    - Used to create external documentation about source code.
    - Documentation generator is the command “javadoc” and it automatically adds all text inside documentation comment to an HTML paragraph.

Java Comments

A. Single-line(c++ style)

- Starts with two forward slash(//) lines and continues to the end of line
  - Eg

    // is single line comment

B. Multiple-line (c-style)

- Starts with /* and ends with */
- Span multiple lines
  - Note: can't be nested

C. Special javadoc comments(documentation comments)

- Used to create external documentation about your source code
- Starts with /** and ends with */
  - Note: can't be nested

Application with two classes

A. Create two classes in the same folder (be it in same file or otherwise)

- Notice the following:
- To access members of the other class
  - **We MUST use object reference with dot operator**
  - **They should not be private (but they can be package private or more)**
- The default access modifier or visibility mode in Java is **PACKAGE PRIVATE**!

B. Create two classes in the different folder

- Notice the following:
- To access members of the other class
  - **We MUST use object reference with dot operator**
  - **The members to be accessed must be declared "Public" (unless there is inheritance)**
  - **The class we are accessing has to be public as well**

Java Program Structure

- Java Program may contain one or more sections and they should appear in the following way:

Documentation Section	Suggested
Package Statement	Optional
Import Statement	”
Interface Statement	”
Class Definitions	”
Main Class Definition (Class with main method)	Essential

Java Identifiers, Keywords and Data Types

A. Java Identifiers

- are names of the things we use in a program (variable, method, class)
- Naming Rules:
  1. Can contain digits (0-9), alphabets (a-z or A-Z), underscore (_), dollar sign ($)
  2. Cannot begin with digit (or a number)
  3. Upper case & lower cases are different (i.e. case sensitive)
  4. Can be of any length
  5. Cannot use Java Keywords.

Java Identifiers, Keywords and Data Types

- **Java Identifiers Coding Guidelines (Conventions)**

1. Names of classes: capitalize leading letter (of each word if it contains more words)

   E.g. HelloJava, Student, MyMotorCycle

2. Names of Methods and Variables: **first letter should be small letter** (all subsequent words should begin with fisrt upper case letter if it contains more words)

   E.g. add, showResult, findAverageMark

3. Avoid using underscore at the start of identifier
4. Constants are ALL CAPS with each word separated by underscore

   E.g. TOTAL_SUM, MAX_NUMBER_OF_STUDENTS

Java Identifiers, Keywords and Data Types

B. Java Keywords

- are predefined identifiers by Java
- They are 50 in number (E.g. int, float, char, break, return, for, do, …)

Java Literals:

- Literals are values which do not change
- Integer Literals: Java Supports the following integer literals
  - Decimal: E.g: -3,-2,-1,0,1,23,…
  - Octal: Preceded by 0 (zero) E.g. 026 (equivalent to 22 in decimal)
  - Hexadecimal: Preceded by 0x (zero & x) E.g. 0x12, 0x1FFB
- Floating Point Literals: can be expressed in standard or scientific form
  - Standard: E.g. 512.34, 45.56, 0.0045
  - Scientific: E.g. 5.1234E2, 4.556e1, 4.5E-3
- Boolean Literals: true, false ONLY
- Character Literals: things in single quote E.g. ‘a’, ‘6’, ‘@’
- String Literals: things in double quote E.g. “a”, “Daniel”, “Good Work!”

Primitive Types

Type	Description	Size	
Boolean (`boolean`)	True/false value	1 bit	true/false (only either of them)
Byte (`byte`)	Byte-length integer	1 byte	[-128 to 127]
Short (`short`)	Short integer	2 bytes	[-32,768 to 32,767]
Integer (`int`)	Integer	4 bytes	[-2,147,483,648 to 2,147,483,647]
Long (`long`)	Long Integer	8 bytes	[-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807]
Float (`float`)	Single precision floating point number	4 bytes	754 floating point
Double (`double`)	Double precision float	8 bytes	754 floating point
Char (`char`)	16-bit Unicode character	2 bytes	'\u0000' (or 0) to '\uffff' (or 65,535 inclusive)

Default Values

when a ***field*** is declared the compiler assigns a default value based on the field data type.

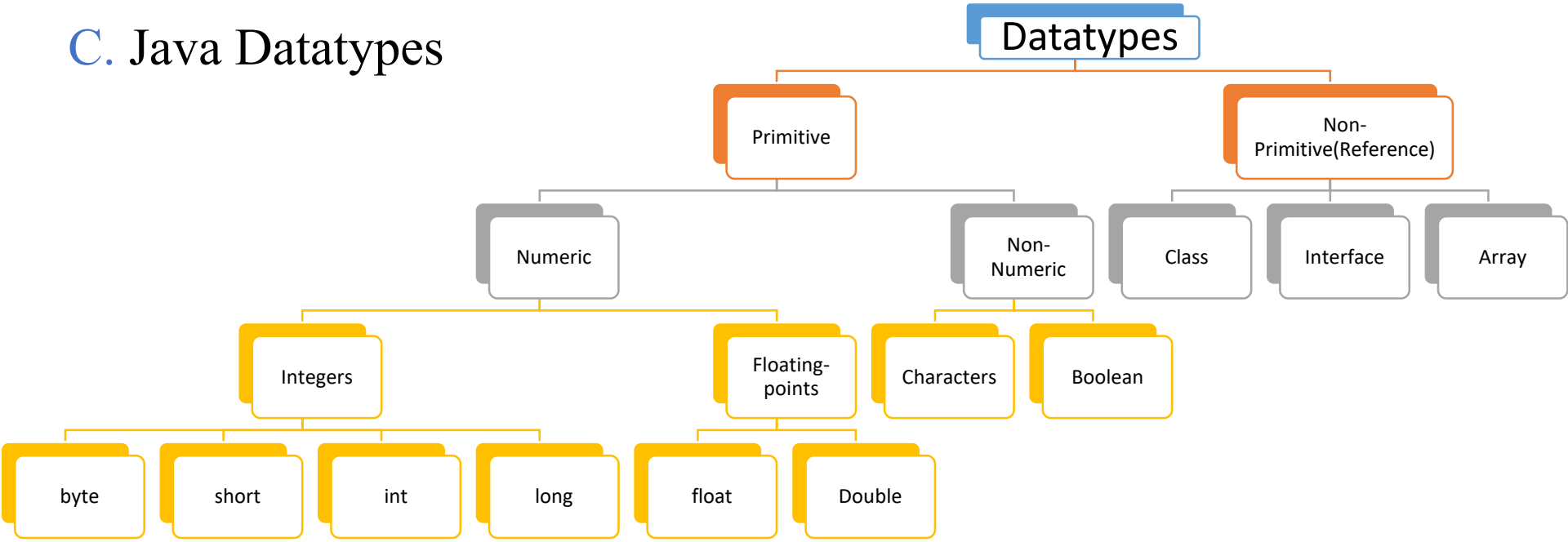
But for ***Local variables*** the compiler never assigns a default value to an uninitialized local variable. If you cannot initialize your local variable where it is declared, make sure to assign it a value before you attempt to use it.

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

- Local variables store temporary state inside a method

Java Identifiers, Keywords and Data Types

C. Java Datatypes



- The primitive types in Java are portable across all computer platforms that support Java. *Eg. In C++* int on one machine might be represented by 16 bits (2 bytes) of memory, on a second machine by 32 bits (4 bytes), and on another machine by 64 bits (8 bytes).
- Primitive types: Refer to actual values
- Reference types: Refer to memory locations (by name, not location)

Scope of Variables (or Type of Variables)

Three types of variables in Java

A. Local Variables

- are declared in methods, constructors, or blocks.
- are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
- Access modifiers cannot be used for local variables.
- are visible only within the declared method, constructor or block.
- are implemented at stack level internally.
- There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use
- Notice:
  - Using uninitialized Local Variable results in error

Java Identifiers, Keywords and Data Types

B. Instance Variables

- are declared in a class, but outside a method, constructor or any block.
- space is allocated for when object is created
- are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object references it is null.
- Instance variables can be accessed directly by calling the variable name inside the class. However within static methods and different class (when instance variables are given accessibility) should be called using the fully qualified name

  ObjectReference.VariableName

Java Identifiers, Keywords and Data Types

C. Class Variables (or Static Variables)

- are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Default values are same as instance variables.
- Static variables can be accessed by calling with the class name or object reference

  ClassName.VariableName. Or *ObjectReference.VariableName*

Operators

A. Arithmetic: (*, /, %, +, -)

- NB:
  - Dividing integer by integer gives integer result by truncating the decimal part if the result is a floating-point
  - Precedence (**/,*,%** are equal and first come first served

    +, - have equal precedence but lesser than the other three)

B. Increment/Decrement (++/--)

C. Relational (**>, <, >=, <=, !=, ==**)

- They return either true or false
- Red ones have higher precedence than blue once

D. Logical Operators (!, &&, ||, &, |)

E. Conditional Operator (or ternary) (**? :**)

Casting

- Is agreeing to a change from one type to another

E.g. float x = 23;

float y = 5.5; //Error, by default floating-points are ‘double’ type

float y = 5.5f; or float y = (float)5.5; // are correct

byte a = 121; //Correct

double m = 8.89;

int n = (int)m; //this assigns 8 to n

Chapter 3

Control Flow Structures and Arrays in Java

Control Flow Structures

- if statements:
  - **if**
  - **if…else**
  - **if…else if**
- switch statement
  - supports integer, characters and strings expressions
  - The labels must be constants
- Loops
  - for
  - while
  - do…while
  - enhanced for
- **continue** and **break** statements

Enhanced for syntax:

```
for(variable : array_name){
    //body
}
```

Enhanced for example

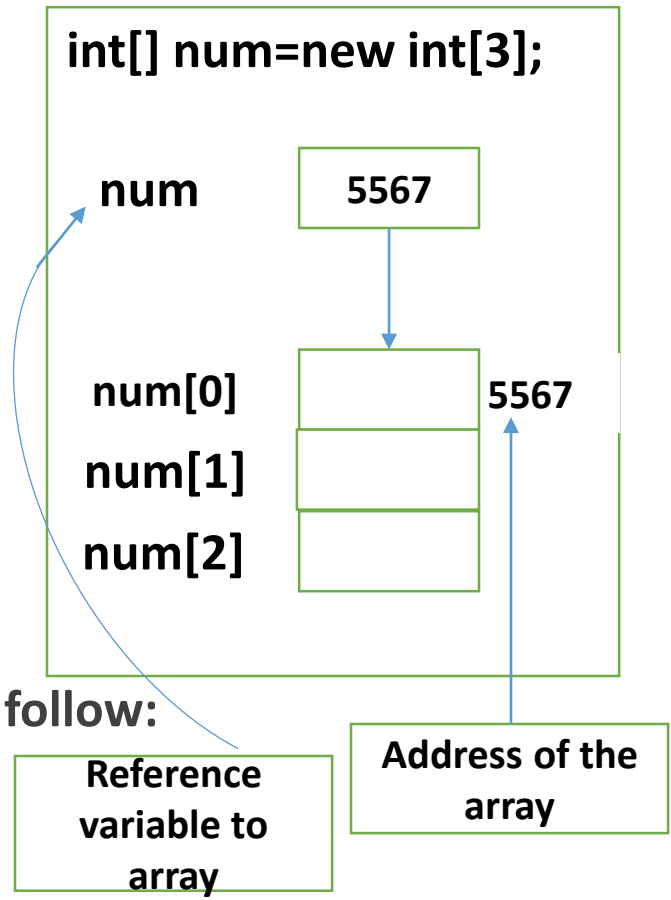
```
int marks[] = {12, 8, 3, 7};

for(int i : marks) {
    S.o.println( i );
}
```

Java Arrays

- Declaring and Creating an array
- Initializing array
- Assigning one array object to another
- Array “length”
- Two Dimensional Array
- Initializing Two Dimensional Array
- Creating irregular size array

Java Arrays

- Array is simply group of data of the same type
- Array is created using "new" keyword implying that it is an object (or a reference type)
  1. **Declaring an Array:**
     - <datatype>[] <arrayReference>;
       E.g. int[] x;
       char[] y;
     - **NB: no mentioning size during declaration**
  2. **Creating an Array:**
     - created using "new" keyword
     - <arrayReference> = new <datatype>[size];
       E.g. x = new int[5];
       y = new char[20];
- **Declaring and Creating can be done in one line as follow:**
  int[] x = new int[5];
  char[] y = new char[20];



Initializing Arrays

- Assigning values to array at the time of declaration

E.g.

```
int[] num = {10,5,4,2,1};
String[] names = {"Dawit", "Helen", "Bonsa"};
double[] marks = {12.56, 6.77, 2.5};
float[] avg = {8.56f, 5.66f, 20.5f, 36.9f};
```

- It is possible to assign one array object to another.

```
int[] a = {1,2,3,4};
int[] b;
b = a;

for(int x : b){
    System.out.println(x);
}
```

output

1
2
3
4

```
int[] x = {1,2,3};
int[] y = new int[6];
y = x;

for(int i : y){
    System.out.println(i);
}
```

output

1
2
3

From this point on 'y' points to the array 'x' is pointing to. But initially 'y' was pointing to array of six elements

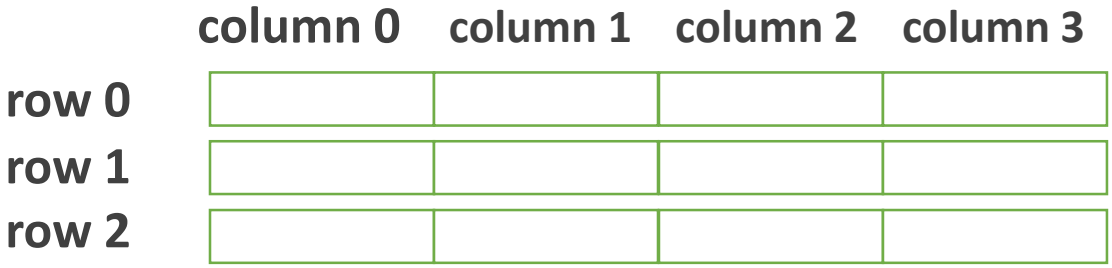
Array Length

- In Java, all arrays store the allocated size in a variable named "length".
- We can obtain length of any array using array reference, dot operator and the 'length' variable.
- Example: Full Program with an array

```
public class ArrayDemo {
      double getAverage(int[] test){
      int sum = 0;
      for(int i=0; i < test.length; i++){
         sum = sum + test[i];
      }
      double avg = (double) sum /test.length;
      return avg;
   }
   public static void main(String[] args) {
      int marks[] = {15,17,13,12,10,20};
      double av;
      ArrayDemo obj = new ArrayDemo();
      av = obj.getAverage(marks);
      System.out.println("Average is: " + av);
   }
}
```

Two-Dimensional Arrays

- Consists of rows and columns
- The 1st index represent rows and the 2nd the columns
- Example: **arr[3][4];**



- To create the above array using code:

```
int[][] arr = new int[3][4];
```

- Initializing two dimensional array:

```
int arr[][] = { {7,7,7} , {8,8,8} }; //Correct
int arr[][] = {7,7,7,8,8,8}; //Error b/c inside braces missing
int arr[2][3] = { {7,7,7} , {8,8,8} }; //Error, why?
```

Two-Dimensional Arrays

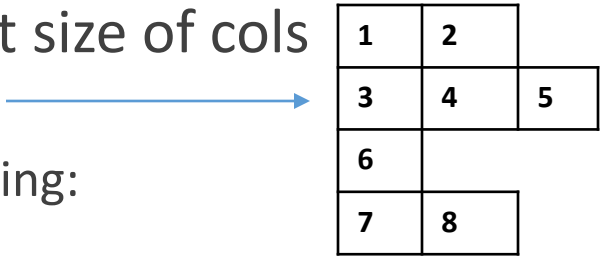
- NB:

  int[][] myArray = new int[4][]; //we are setting up 4 rowed array

  int[][] myArray = new int[][4]; //Error, number of rows not mentioned

- Variable size array: array having different size of cols

  int a[][] = {{1,2}, {3,4,5}, {6}, {7,8}};



- Creating above array using code without initializing:

  int a[][] = new int[4][]; //row size is fixed

  a[0] = new int[2]; //row 0 itself is a single dimensional array of size 2

  a[1] = new int[3]; //row 1 itself is a single dimensional array of size 3

  a[2] = new int[1]; //row 2 itself is a single dimensional array of size 1

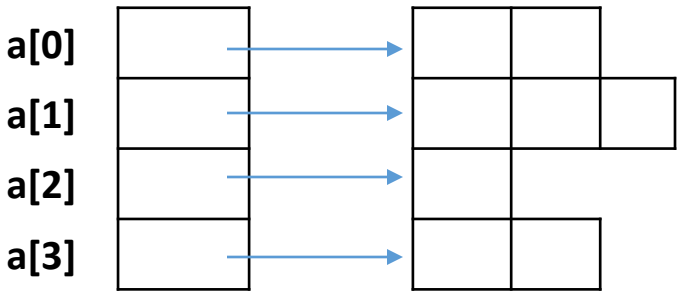
  a[3] = new int[2]; //row 3 itself is a single dimensional array of size 2

- The above code creates the following:
  - ✓ We simply created the references
  - ✓ Then we can add elements:

    a[0][0] = 1;

    a[2][0] = 6;

    a[1][2] = 5; etc



Two-Dimensional Arrays

- Example:
  - The following function takes two dimensional array and display each element.

```
void show(float[][] myArray)
{
      for(int i = 0; i < myArray.length; i++){
            for(int b = 0; b < myArray[i].length; b++){
                  System.out.print( myArray[i][b] + "\t");
            }
             System.out.println(); //just to insert new line for the next row
      }
}
```

Variable Length Arguments (VARARGS)

- We can create methods that receive an unspecified number of argumetns.
- We use ellipses (…) in the methods parameters list next to argument type.
- NB:
  - Use of ellipses can occur **only once** in parameters list
  - AND must be placed **at the end** of parameter list

```
void avgFun(double… marks){
    double total = 0;
    for(int m : marks) {
        total = total + m;
    }
    System.out.println("Average: " +
                        total/marks.length);
}
```

We can call the function with any number of arguments:

avgFun(3,4,5); // OR
avgFun(4.5,6,7,2.5,7); //OR
avgFun(3);
avgFun(); //possible but the marks.length is zero & creates exception

Chapter 4
Classes and Objects

Classes and Objects

- Classes
  - Defining a class – Syntax
  - Declaring Member Variables
  - Class Members Access Modifiers (Visibility Labels or Scopes)(Just Introduction)
  - Defining Methods (Method Signature)
  - Overloading Methods
- Objects
  - Creating Object (Instantiating a class)
  - Reference vs Actual Object
- Accessing Class Members
  - If inside a class
  - If outside a class (from other classes)
- Memory Allocation for Objects

Classes and Objects

- Static Data Members (In relation to memory allocation)
- Java Variable Types
  - Local
  - Instance
  - Static (or Class)
- Static Methods
- Controlling Access to Members of a class
  - Access Modifiers (Visibility Labels or Scopes) (In detail)
- Arrays as member of class
- Arrays of Objects
- Objects as Function Argument
- Returning Objects from Methods (Functions)

Classes and Objects

- Accessors and Mutators Methods
- Static Initialization Blocks
- Initializer Block (to mean instance initialization block)
- Nested Classes
  - Static Nested Class
  - Inner Classes
    - Local Classes
    - Anonymous Classes

Class

- is the blueprint from which individual objects are created
- is user defined data type (i.e. once class is defined we can create variables of that class type)
- binds DATA and its associated FUNCTIONS, i.e. it contain data items and functions.
  - The data items are called **Fields**
  - The functions are called **Methods**
- Fields and Methods are collectively called **Class Members.**

Defining a Class

- Syntax:

```
[public] class <className> <extends> <parentClassName> <implements> <interfaceName>
{
      //body
}
```

- ..

Java Comments

- **Comments**: are notes written in the source code by the programmer to make the code easier to understand.
  - Comments are ignored by the compiler but are useful to other programmers.
  - Comments are not executed when your program runs.
  - Most Java editors show your comments with a special color.
- **Java supports three kinds of comments:**
  - **Single-Line Comment - //**
  - **Multiple-Line Comment - /* ……. */**
  - **Documentation Comment (Javadoc Comment) - /**……. */**
    - Used to create external documentation about source code.
    - Documentation generator is the command “javadoc” and it automatically adds all text inside documentation comment to an HTML paragraph.

Java Comments

- **Comments**: are notes written in the source code by the programmer to make the code easier to understand.
  - Comments are ignored by the compiler but are useful to other programmers.
  - Comments are not executed when your program runs.
  - Most Java editors show your comments with a special color.
- **Java supports three kinds of comments:**
  - **Single-Line Comment - //**
  - **Multiple-Line Comment - /* ……. */**
  - **Documentation Comment (Javadoc Comment) - /**……. */**
    - Used to create external documentation about source code.
    - Documentation generator is the command “javadoc” and it automatically adds all text inside documentation comment to an HTML paragraph.

Chapter 5
Constructors

Constructors

- Defining Constructors
- Rules of using Constructors
- Default Constructors
- Parameterized Constructors
- Overloaded Constructors
- Constructors in Inheritance

Chapter 6
Inheritance & Interfaces

Inheritance & Interfaces

- Concept of inheritance
- Super classes and subclasses
- Protected members
- Constructors in subclasses
- Casting Objects
- Overriding and Hiding methods
- Using **this**() and **super**()
- Final Classes and Final Methods
- Abstract Classes and Abstract Methods
- Interfaces

Chapter 7
Polymorphism

Polymorphism

- ....

Chapter 8
Exception Handling

Exception Handling

- Exception handling overview
- The causes of exceptions
- The Throwable class hierarchy
- Handling of an exception
- The throw statement
- The finally clause
- User defined exceptions