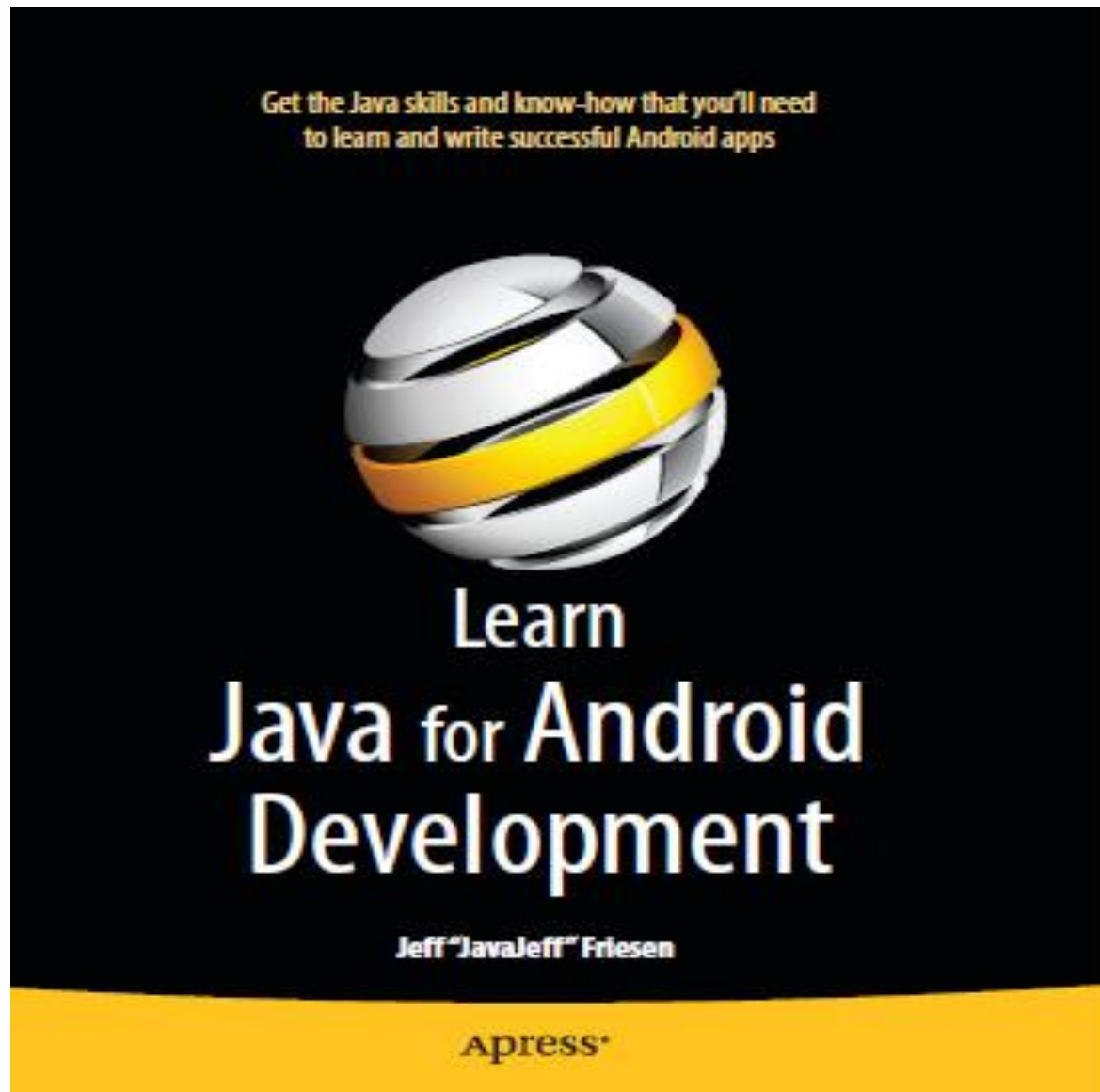




Java Programming for android

Dieudonne U

Book to read





Java Pre-requisite

- **Getting Started with Java**
- **Learning Language Fundamentals**
- **Learning Object-Oriented Language Features**



Focus of the Course

- Object-Oriented Software Development
 - program design, implementation
 - object-oriented concepts
 - classes
 - objects
 - encapsulation
 - inheritance
 - Polymorphism
 - problem solving



Chapter 1: Introduction

- *Android is Google's software stack for mobile devices*
- *Includes an operating system and middleware.*
- *With help from Java, the OS runs specially designed Java applications*
 - *known as Android apps.*
- *Because these apps are based on Java, it makes sense for you*

What is Java

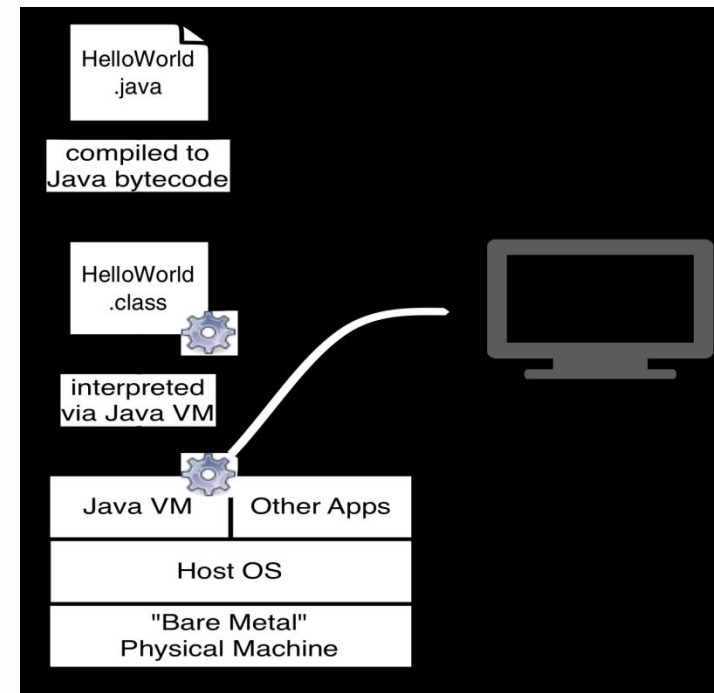
- *Java is a language and a platform originated by Sun Microsystems.*
- It was introduced in 1995 and its popularity has grown quickly since
- Sun organized Java into three main editions: Java SE, Java EE, and Java ME.
- Similarities with C/C++ or C#
- Java is a platform for executing programs.
 - In contrast to platforms that consist of physical processors (such as an Intel processor) and operating systems (such as Linux),

What is Java

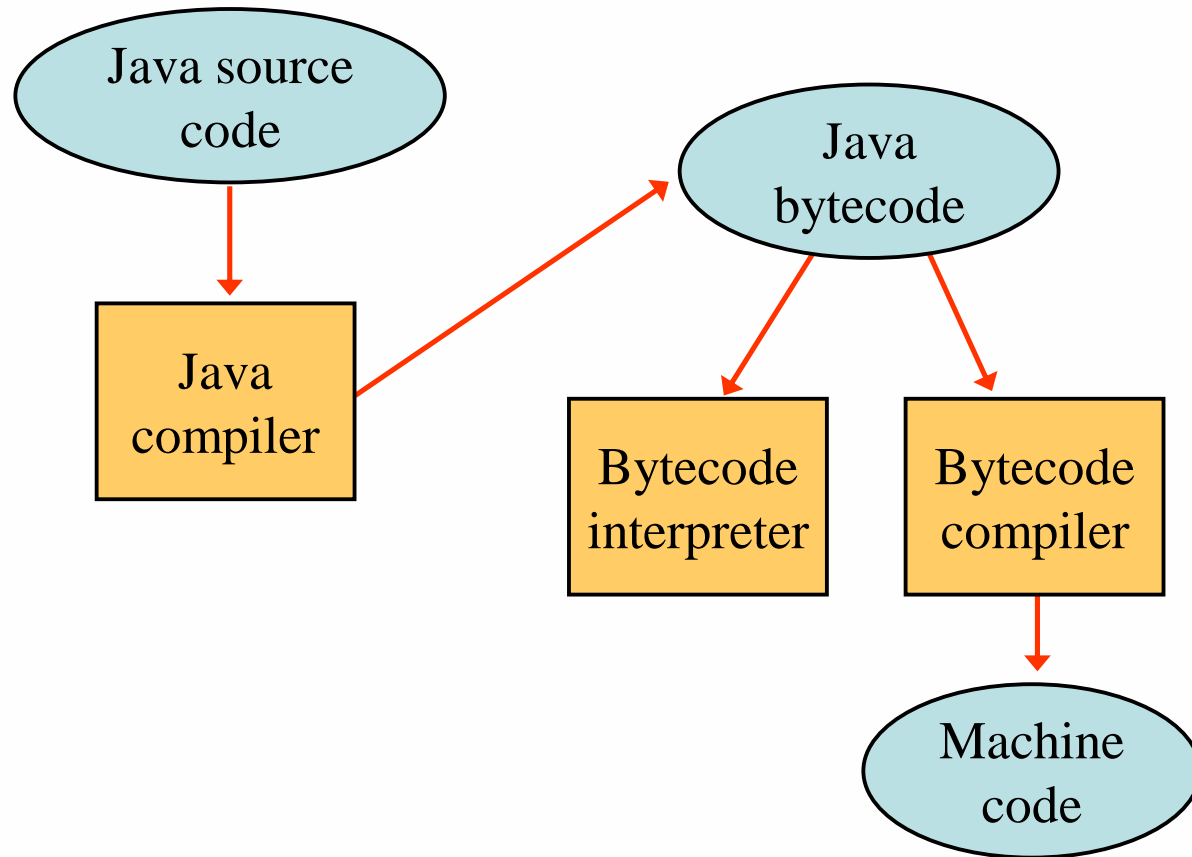
- The Java platform consists of a virtual machine and associated execution environment.
- The *virtual machine is a software-based processor that presents its own instruction set.*
- The associated *execution environment consists of libraries for running programs and* interacting with the underlying operating system.
- The execution environment includes a huge library of prebuilt class files referred to as the *standard class library.*
- A special Java program known as the *Java compiler translates source code into* instructions (and associated data) that are executed by the virtual machine.

Java Programming Language

- Java: general-purpose language designed so developers write code once, it runs anywhere
- The key: Java Virtual Machine (JVM)
 - Program code compiled to JVM bytecode
 - JVM bytecode interpreted on JVM



Java Translation



Java SE, Java EE, Java ME, and Android

Different editions of the Java platform to create Java programs that run

- on desktop computers, web browsers, web servers, mobile information devices (such as cell phones), and embedded devices (such as television set-top boxes):
- Java Platform, Standard Edition (Java SE): *The Java platform for developing applications, which are stand-alone programs that run on Desktops.*
- Java Platform, Enterprise Edition (Java EE): *The Java platform for developing enterprise-oriented applications*
- Java Platform, Micro Edition (Java ME): *The Java platform for developing MIDlets, which are programs that run on mobile information devices and on embedded devices.*



Java Program Structure

- In the Java programming language:
 - A program is made up of one or more *classes*
 - A class contains one or more *methods*
 - A method contains program *statements*
- These terms will be explored in detail throughout the course
- A Java application always contains a method called `main`

Lincoln.java

```
public class Lincoln
{
    //-----
    // Prints a presidential quote.
    //-----
    public static void main (String[] args)
    {
        System.out.println ("A quote by Abraham Lincoln:");

        System.out.println ("Whatever you are, be a good one.");
    }
}
```

Java Program Structure

```
// comments about the class
```

```
public class MyProgram
```

```
{
```

class header

class body

Comments can be placed almost anywhere

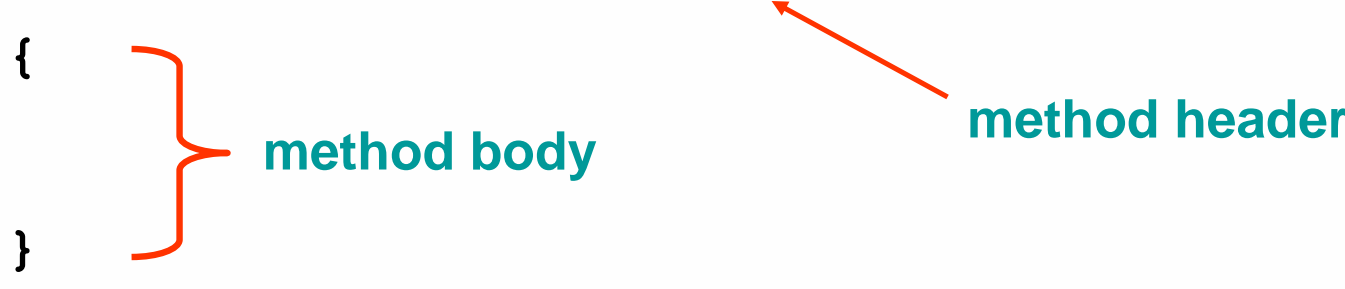
```
}
```

The class MyProgram must have the same name as the source file MyProgram.java -
Our class has public

Java Program Structure

```
// comments about the class
public class MyProgram
{
    // comments about the method
    public static void main (String[] args)
    {
        }
}

method body
method header
```



Comments

- Comments in a program are called *inline documentation*
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take three forms:

```
// this comment runs to the end of the line
```

```
/*  this comment runs to the terminating  
    symbol, even across line breaks      */
```

```
/** this is a javadoc comment    */
```


Identifiers

- ***Identifiers*** are the words a programmer uses in a program
- An identifier can be made up of letters, digits, the underscore character (`_`), and the dollar sign
- Identifiers cannot begin with a digit
- Java is ***case sensitive*** - `Total`, `total`, and `TOTAL` are different identifiers
- By convention, programmers use different case styles for different types of identifiers, such as
 - ***title case*** for class names - `Lincoln`
 - ***upper case*** for constants - `MAXIMUM`

Reserved Words

- The Java reserved words:

<code>abstract</code>	<code>else</code>	<code>interface</code>	<code>switch</code>
<code>assert</code>	<code>enum</code>	<code>long</code>	<code>synchronized</code>
<code>boolean</code>	<code>extends</code>	<code>native</code>	<code>this</code>
<code>break</code>	<code>false</code>	<code>new</code>	<code>throw</code>
<code>byte</code>	<code>final</code>	<code>null</code>	<code>throws</code>
<code>case</code>	<code>finally</code>	<code>package</code>	<code>transient</code>
<code>catch</code>	<code>float</code>	<code>private</code>	<code>true</code>
<code>char</code>	<code>for</code>	<code>protected</code>	<code>try</code>
<code>class</code>	<code>goto</code>	<code>public</code>	<code>void</code>
<code>const</code>	<code>if</code>	<code>return</code>	<code>volatile</code>
<code>continue</code>	<code>implements</code>	<code>short</code>	<code>while</code>
<code>default</code>	<code>import</code>	<code>static</code>	
<code>do</code>	<code>instanceof</code>	<code>strictfp</code>	
<code>double</code>	<code>int</code>	<code>super</code>	



White Space

- Spaces, blank lines, and tabs are called *white space*
- White space is used to separate words and symbols in a program
- Extra white space is ignored
- A valid Java program can be formatted many ways
- Programs should be formatted to enhance readability, using consistent indentation



Lincoln2.java

```
public class Lincoln2{public static void  
    main(String[]args){  
    System.out.println("A quote by Abraham Lincoln:");  
    System.out.println("Whatever you are, be a good  
        one.");}}
```



Basic Datatypes

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in the memory.
- Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.
- There are two data types available in Java –
 - Primitive Data Types
 - Reference/Object Data Types

Java's primitive data types:

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	1-bit	N/A	N/A	Boolean
char	16-bit	Unicode 0	Unicode $2^{16} - 1$	Character
byte	8-bit	-128	+127	Byte
short	16-bit	-2^{15}	$+2^{15} - 1$	Short
int	32-bit	-2^{31}	$+2^{31} - 1$	Integer
long	64-bit	-2^{63}	$+2^{63} - 1$	Long
float	32-bit	IEEE 754	IEEE 754	Float
double	64-bit	IEEE 754	IEEE 754	Double

Reference Datatypes

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Student, Employee, Puppy, etc.
- Class objects and various type of array variables come under reference datatype.
- Default value of any reference variable is null.
- A reference variable can be used to refer any object of the declared type or any compatible type.
 - **Example: `Student newStudent = new Student ("Eric");`**

Variable Types

- A variable provides us with named storage that our programs can manipulate.
- Each variable in Java has a specific type, which determines the size and layout of the variable's memory;
- The range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
- You must declare all variables before they can be used. Following is the basic form of a variable declaration
 - **int a, b, c; // Declares three ints, a, b, and c.**



Variable Types

- There are three kinds of variables in Java
 - Local variables
 - Instance variables
 - Class/Static variables

Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables 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.
- Local variables are visible only within the declared method, constructor, or block. 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.

```
public class Student {  
    public void StudentAge() {  
        int age = 0; // local variable, needs to be initialized  
        age = age + 7;  
        System.out.println("Puppy age is : " + age);  
    }  
    public static void main(String args[]) {  
        Student student = new Student();  
        student.StudentAge();  
    }  
}
```

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.

Instance Variables cont'd

```
import java.io.*;
public class Employee {
    // this instance variable is visible for any child class.
    public String name;

    // salary variable is visible in Employee class only.
    private double salary;

    // The name variable is assigned in the constructor.
    public Employee (String empName) {
        name = empName;
    }

    // The salary variable is assigned a value.
    public void setSalary(double empSal) {
        salary = empSal;
    }

    // This method prints the employee details.
    public void printEmp() {
        System.out.println("name : " + name );
        System.out.println("salary :" + salary);
    }

    public static void main(String args[]) {
        Employee empOne = new Employee("Ransika");
        empOne.setSalary(1000);
        empOne.printEmp();
    }
}
```

Class/Static Variables

- Class variables also known as 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. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.

```
import java.io.*; public class Employee {  
    // salary variable is a private static variable  
    private static double salary;  
    // DEPARTMENT is a constant  
    public static final String DEPARTMENT = "Development ";  
    public static void main(String args[]) {  
        salary = 1000; System.out.println(DEPARTMENT + "average salary:" + salary);  
    }  
}
```


Access modifiers

- Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are –
 - Visible to the package, the default. No modifiers are needed.
 - Visible to the class only (private).
 - Visible to the world (public).
 - Visible to the package and all subclasses (protected).

```
public class className { // ... }  
private boolean myFlag;  
static final double weeks = 9.5;  
protected static final int BOXWIDTH = 42;  
public static void main(String[] arguments) {  
    // body of method  
}
```




Basic Operators

- Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups –
 - Arithmetic Operators
 - Relational Operators
 - Bitwise Operators
 - Logical Operators
 - Assignment Operators
 - Misc Operators

Loop Control - **while**

- A **while** loop statement in Java programming language repeatedly executes a target statement as long as a given condition is true.

```
public class Test {  
    public static void main(String args[]) {  
        int x = 10;  
        while( x < 20 ) {  
            System.out.print("value of x : " + x );  
            x++;  
            System.out.print("\n");  
        }  
    }  
}
```

Loop Control - **for**

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times.

```
public class Test {  
    public static void main(String args[]) {  
        for(int x = 10; x < 20; x = x + 1) {  
            System.out.print("value of x : " + x ); System.out.print("\n");  
        }  
    }  
}
```

Loop Control cont'd - do...while

- A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

```
public class Test {  
    public static void main(String args[]) {  
        int x = 10;  
  
        do {  
            System.out.print("value of x : " + x );  
            x++;  
            System.out.print("\n"); }  
        while( x < 20 );  
    }  
}
```

Decision Making

- Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

```
if(Boolean_expression) {  
    // Statements will execute if the Boolean expression is true  
}
```

```
switch(expression) {  
    case value :  
        // Statements  
        break;  
    // optional  
    case value :  
        // Statements  
        break;  
    // optional // You can have any number of case statements.  
    default : // Optional // Statements  
}
```



Methods

- A Java method is a collection of statements that are grouped together to perform an operation.
- When you call the **System.out.println()** method, for example, the system actually executes several statements in order to display a message on the console.
- You can create your own methods with or without return values, invoke a method with or without parameters, and apply method abstraction in the program design.

Creating Method

```
public static int methodName(int a, int b) {  
    // body  
}
```

public static – modifier: It defines the access type of the method and it is optional to use.

int – returnType: Method may return a value.

methodName – name of the method: This is the method name. The method signature consists of the method name and the parameter list.

int a, int b – list of parameters, **a, b** – formal parameters

Method definition consists of a method header and a method body.

method body – The method body defines what the method does with the statements.

Method Calling

- For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).
- The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method.

```
public class ExampleMinNumber {
    public static void main(String[] args) {
        int a = 11;
        int b = 6;
        int c = minFunction(a, b);
        System.out.println("Minimum Value = " + c);
    }
    /** returns the minimum of two numbers */
    public static int minFunction(int n1, int n2) {
        int min;
        if (n1 > n2) { min = n2 ;}
        else {min = n1; }

        return min;
    }
}
```

Method - The void Keyword

- The void keyword allows us to create methods which do not return a value.
- Here, in the following example we're considering a void method *methodRankPoints*. This method is a void method, which does not return any value

```
public class ExampleVoid {  
    public static void main(String[] args) {  
        methodRankPoints(255.7);  
    }  
    public static void methodRankPoints(double points) {  
        if (points >= 202.5) {  
            System.out.println("Rank:A1");  
        }  
        else if (points >= 122.4) {  
            System.out.println("Rank:A2");  
        }  
        else {  
            System.out.println("Rank:A3");  
        }  
    }  
}
```

Method - Passing Parameters by Value

- While working under calling process, arguments is to be passed.
- These should be in the same order as their respective parameters in the method specification.
- Parameters can be passed by value or by reference.
- Passing Parameters by Value means calling a method with a parameter.

```
public class swappingExample {  
    public static void main(String[] args) {  
        int a = 30; int b = 45;  
        System.out.println("Before swapping, a = " + a + " and b = " + b);  
  
        // Invoke the swap method  
        swapFunction(a, b);  
        System.out.println("\n**Now, Before and After swapping values will be same here**");  
        System.out.println("After swapping, a = " + a + " and b is " + b);  
    }  
  
    public static void swapFunction(int a, int b) {  
        System.out.println("Before swapping(Inside), a = " + a + " b = " + b);  
  
        // Swap n1 with n2  
        int c = a; a = b; b = c;  
        System.out.println("After swapping(Inside), a = " + a + " b = " + b);  
    }  
}
```

Method Overloading

- When a class has two or more methods by the same name but different parameters, it is known as method overloading.
- It is different from overriding. In overriding, a method has the same method name, type, number of parameters, etc.
- Let's consider the example discussed earlier for finding minimum numbers of integer type.
 - If, let's say we want to find the minimum number of double type.
 - Then the concept of overloading will be introduced to create two or more methods with the same name but different parameters.

Method Overloading cont'd

```
public class ExampleOverloading {
    public static void main(String[] args) {
        int a = 11; int b = 6; double c = 7.3; double d = 9.4;
        int result1 = minFunction(a, b);

        // same function name with different parameters
        double result2 = minFunction(c, d);
        System.out.println("Minimum Value = " + result1);
        System.out.println("Minimum Value = " + result2);
    }

    // for integer
    public static int minFunction(int n1, int n2) {
        int min;
        if (n1 > n2) min = n2;
        else min = n1;
        return min;
    }

    // for double
    public static double minFunction(double n1, double n2) {
        double min; if (n1 > n2) min = n2;
        else min = n1;
        return min;
    }
}
```

The Constructors

- A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.
- Typically, you will use a constructor to give initial values to the instance variables defined by the class, or to perform any other startup procedures required to create a fully formed object.
- All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero. However, once you define your own constructor, the default constructor is no longer used.

```
// A simple constructor.  
class MyClass {  
    int x;  
    // Following is the constructor  
    MyClass() {  
        x = 10;  
    }  
}
```


Parameterized Constructor

- Most often, you will need a constructor that accepts one or more parameters.
- Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.

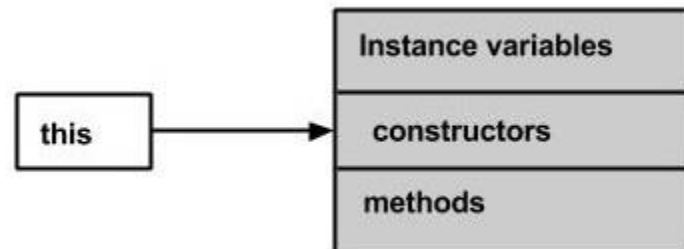
// A simple constructor.

```
class MyClass {  
    int x;  
    // Following is the constructor  
    MyClass(int i) {  
        x = i;  
    }  
}
```


The this keyword

- **this** is a keyword in Java which is used as a reference to the object of the current class, with in an instance method or a constructor.
- Using *this* you can refer the members of a class such as constructors, variables and methods.
- The keyword *this* is used only within instance methods or constructors

```
class Student {  
    int age;  
    Student(int age) {  
        this.age = age;  
    }  
}
```



Date and Time

- Java provides the **Date** class available in **java.util** package, this class encapsulates the current date and time.
- The Date class supports two constructors as shown in the following table.
- Getting Current Date and Time

```
import java.util.Date;
public class DateDemo {
    public static void main(String args[]) {
        // Instantiate a Date object
        Date date = new Date();
        // display time and date using toString()
        System.out.println(date.toString());
    }
}
```

Date Formatting Using SimpleDateFormat

- SimpleDateFormat is a concrete class for formatting and parsing dates in a locale-sensitive manner. SimpleDateFormat allows you to start by choosing any user-defined patterns for date-time formatting.

```
import java.util.*;
import java.text.*;
public class DateDemo {
    public static void main(String args[]) {
        Date dNow = new Date( );
        SimpleDateFormat ft = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
        System.out.println("Current Date: " + ft.format(dNow));
    }
}
```

Parsing Strings into Dates

- The SimpleDateFormat class has some additional methods, notably parse(), which tries to parse a string according to the format stored in the given SimpleDateFormat object.

```
import java.util.*;
import java.text.*;
public class DateDemo {
    public static void main(String args[]) {
        SimpleDateFormat ft = new SimpleDateFormat ("yyyy-MM-dd");
        String input = args.length == 0 ? "1818-11-11" : args[0];
        System.out.print(input + " Parses as ");
        Date t;
        try {
            t = ft.parse(input);
            System.out.println(t);
        }
        catch (ParseException e) {
            System.out.println("Unparseable using " + ft);
        }
    }
}
```

Inner classes

- In Java, just like methods, variables of a class too can have another class as its member. Writing a class within another is allowed in Java. The class written within is called the **nested class**, and the class that holds the inner class is called the **outer class**.
- Creating an inner class is quite simple. You just need to write a class within a class. Unlike a class, an inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside the class.

```
class Outer_Demo {
    int num;
    // inner class
    private class Inner_Demo {
        public void print() {
            System.out.println("This is an inner class");
        }
    }
    // Accessing the inner class from the method within
    void display_Inner() {
        Inner_Demo inner = new Inner_Demo();
        inner.print();
    }
}

public class My_class {
    public static void main(String args[]) {
        // Instantiating the outer class
        Outer_Demo outer = new Outer_Demo();
        // Accessing the display_Inner() method.
        Outer.display_Inner();
    }
}
```

Exceptions

- An exception (or exceptional event) is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled
 - An exception can occur for many different reasons. Following are some scenarios where an exception occurs.
 - A user has entered an invalid data.
 - A file that needs to be opened cannot be found.
 - A network connection has been lost in the middle of communications or the JVM has run out of memory.
- Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Catching Exceptions

- A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code
- The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception occurred is handled by catch block associated with it. Every try block should be immediately followed either by a catch block or finally block.
- A catch statement involves declaring the type of exception you are trying to catch.

// File Name : ExceptTest.java

```
import java.io.*;
```

```
public class ExceptTest {
```

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

```
        try {
```

```
            int a[] = new int[2];
```

```
            System.out.println("Access element three :" + a[3]);
```

```
        }
```

```
        //use catch (Exception e) for general error excep.
```

```
        catch(ArrayIndexOutOfBoundsException e) {
```

```
            System.out.println("Exception thrown :" + e);
```

```
        }
```

```
        System.out.println("Out of the block");
```

```
    }
```