JAVA Notes-part1- 4CME

Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented. It is fast, reliable, and secure.

1. Java is platform independent

   Java was built with the philosophy of "write once, run anywhere" (WORA). The Java code you write on one platform (operating system) will run on other platforms with no modification.

   To run Java, an abstract machine called Java Virtual Machine (JVM) is used. The JVM executes the Java bytecode. Then, the CPU executes the JVM. Since all JVMs works exactly the same, the same code works on other operating systems as well, making Java platform-independent.

2. An object-oriented Language

   There are different styles of programming. Object-oriented approach is one of the popular programming styles. In object-oriented programming, a complex problem is divided into smaller sets by creating objects. This makes your code reusable, has design benefits, and makes code easier to maintain.

   Many programming languages including Java, Python, and C++ has object-oriented features.

3. Java is fast

   Java is one of the fastest programming languages. Well optimized Java code is nearly as fast as lower level languages like C/C++, and much faster than Python, PHP etc.

4. Java is secure

The Java platform provides various features for security of Java applications. Some of the high-level features that Java handles are:

- provides secure platform for developing and running applications
- automatic memory management, reduces memory corruption and vulnerabilities
- provides secure communication by protecting the integrity and privacy of data transmitted

5. Large Standard Library

One of the reasons why Java is widely used is because of the availability of huge standard library. The Java environment has hundreds of classes and methods under different packages to help software developers like us. For example,

`java.lang` - for advanced features of strings, arrays etc.
`java.util` - for data structures, regular expressions, date and time functions etc.
`java.io` - for file i/o, exception handling etc.

Object oriented programming:

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Applications of using OOP:
Main application areas of OOP are

- User interface design such as windows, menu ,…
- Real Time Systems

- Simulation and Modeling
- Object oriented databases
- AI and Expert System
- Neural Networks and parallel programming
- Decision support and office automation system etc

Benefits of OOP

The main advantages are

- It is easy to model a real system as real objects are represented by programming objects in OOP. The objects are processed by their member data and functions. It is easy to analyze the user requirements.
- With the help of inheritance, we can reuse the existing class to derive a new class such that the redundant code is eliminated and the use of existing class is extended. This saves time and cost of program.
- In OOP, data can be made private to a class such that only member functions of the class can access the data. This principle of data hiding helps the programmer to build a secure program that can not be invaded by code in other part of the program.
- With the help of polymorphism, the same function or same operator can be used for different purposes. This helps to manage software complexity easily.
- Large problems can be reduced to smaller and more manageable problems. It is easy to partition the work in a project based on objects.
- It is possible to have multiple instances of an object to co-exist without any interference i.e. each object has its own separate member data and function.

Introduction to Java

Structure of Java Program

When we consider a Java program, it can be defined as a collection of objects that communicate via invoking each other's methods.

- Object − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.

- Class − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.
- Methods − A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- Instance Variables − Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Java Variables

Following are the types of variables in Java −

- Local Variables
- Class Variables (Static Variables)
- Instance Variables (Non-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.

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.
- Static variables are created when the program starts and destroyed when the program stops.

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.

Comments in Java

Java supports single-line and multi-line comments very similar to C and C++. All characters available inside any comment are ignored by Java compiler.

Example

```
public class MyFirstJavaProgram {


  /* This is my first java program.
   * This will print 'Hello World' as the output
   * This is an example of multi-line comments.
```

```
   */


  public static void main(String []args) {
     // This is an example of single line comment
     /* This is also an example of single line comment. */
     System.out.println("Hello World");
  }
}
```

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts −

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method

Objects in Java

Let us now look deep into what are objects. If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior.

If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running.

If you compare the software object with a real-world object, they have very similar characteristics.

Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.

So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

Classes in Java

A class is a blueprint from which individual objects are created.

Following is a sample of a class.

Example

```
public class Dog {
   String breed;
   int age;
   String color;

   void barking() {
   }

   void hungry() {
   }

   void sleeping() {
   }
}
```

A class can contain any of the following variable types.

- Local variables − Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- Instance variables − Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

- Class variables − Class variables are variables declared within a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

Constructors

Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Following is an example of a constructor −

Example

```
public class Puppy {
  public Puppy() {
  }

  public Puppy(String name) {
    // This constructor has one parameter, name.
  }
}
```

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 start-up 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.

Java allows two types of constructors namely −

- No argument Constructors
- Parameterized Constructors

No argument Constructors

As the name specifies the no argument constructors of Java does not accept any parameters instead, using these constructors the instance variables of a method will be initialized with fixed values for all objects.

Example

```
Public class MyClass {
  Int num;
  MyClass() {
    num = 100;
  }
}
```

Parameterized Constructors

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.

Example

Here is a simple example that uses a constructor −

```
// A simple constructor.
class MyClass {
  int x;

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

There are two data types available in Java −

- Primitive Data Types
- Reference/Object Data Types
-

Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

Creating an Object

As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class −

- Declaration − A variable declaration with a variable name with an object type.
- Instantiation − The 'new' keyword is used to create the object.
- Initialization − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Constants

Constants in java are fixed values those are not changed during the Execution of program java supports several types of Constants those are :

1. Integer Constants
2. Real Constants
3. Single Character Constants
4. String Constants

Operators

There are many types of operators in java which are given below:

- Unary Operator,
- Arithmetic Operator,

- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Introduction to Arrays

an array is a collection of similar type of elements that have a contiguous memory location.

Java array is an object which contains elements of a similar data type. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in java is index-based, the first element of the array is stored at the 0 index.

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Creating Arrays

You can create an array by using the new operator with the following syntax −

Syntax

```
arrayRefVar = new dataType[arraySize];
```

The above statement does two things −

- It creates an array using new dataType[arraySize].
- It assigns the reference of the newly created array to the variable arrayRefVar.

foreach Loops

JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

Example

The following code displays all the elements in the array myList −

```
public class TestArray {
```

```
  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};

    // Print all the array elements
    for (double element: myList) {
      System.out.println(element);
    }
  }
}
```

This will produce the following result −

Output

```
1.9
2.9
3.4
3.5
```

Passing Arrays to Methods

Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an **int** array −

Example

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

You can invoke it by passing an array. For example, the following statement invokes the printArray method to display 3, 1, 2, 6, 4, and 2 −

Example

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Returning an Array from a Method

A method may also return an array. For example, the following method returns an array that is the reversal of another array −

Example

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
    result[j] = list[i];
  }
  return result;
}
```