COSC INTERNSHIP DRIVE

CORE & OBJECT ORIENTED JAVA

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

Why to Learn java Programming?

Java is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Software Development Domain. Some of the key advantages of learning Java Programming:

Object Oriented − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

Platform Independent − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

Simple − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

Secure − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

Architecture-neutral − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

Portable − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable.

Robust − Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

Hello World using Java Programming.

```
public class MyFirstJavaProgram {

  public static void main(String []args) {

    System.out.println("Hello World"); // prints Hello World

  }

}
```

Local Environment Setup

If you are still willing to set up your environment for Java programming language, then this section guides you on how to download and set up Java on your machine. Following are the steps to set up the environment.

Java SE is freely available from the link Download Java. You can download a version based on your operating system.

Follow the instructions to download Java and run the **.exe** to install Java on your machine. Once you installed Java on your machine, you will need to set environment variables to point to correct installation directories −

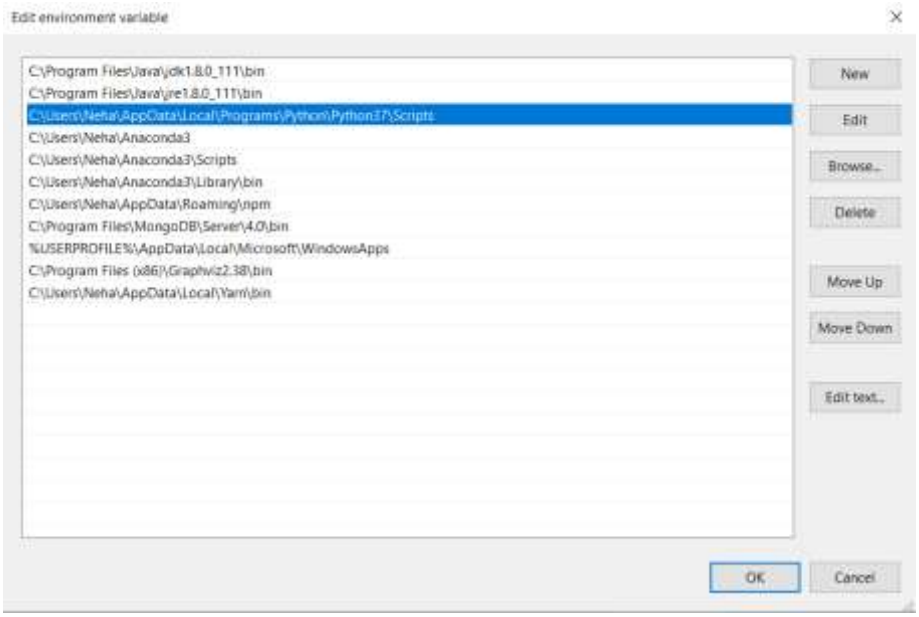
Setting Up the Path for Windows

Assuming you have installed Java in *c:\Program Files\java\jdk* directory −

Right-click on 'My Computer' and select 'Properties'.

Click the 'Environment variables' button under the 'Advanced' tab.

Now, alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.



It should look something like in the first 2 lines in the above picture, the path might change based on your account so please be careful while setting it up.

Setting Up the Path for Linux, UNIX.

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation, if you have trouble doing this.

Example, if you use *bash* as your shell, then you would add the following line to the end of your '.bashrc: export PATH = /path/to/java:$PATH'

Popular Java Editors

To write your Java programs, you will need a text editor. There are even more sophisticated IDEs available in the market. But for now, you can consider one of the following −

Notepad − On Windows machine, you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.

Netbeans − A Java IDE that is open-source and free which can be downloaded from https://www.netbeans.org/index.html.

Eclipse − A Java IDE developed by the eclipse open-source community and can be downloaded from https://www.eclipse.org/.

Basic Syntax

About Java programs, it is very important to keep in mind the following points.

Case Sensitivity - Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.

Class Names − For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

Example: *class MyFirstJavaClass*

Method Names − All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

Example: *public void myMethodName()*

Program File Name − Name of the program file should exactly match the class name.

When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).

But please make a note that in case you do not have a public class present in the file then file name can be different than class name. It is also not mandatory to have a public class in the file.

Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*

public static void main(String args[]) − Java program processing starts from the main() method which is a mandatory part of every Java program.

Java Identifiers

All Java components require names. Names used for classes, variables, and methods are called **identifiers**.

In Java, there are several points to remember about identifiers. They are as follows −

- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- A key word cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.
- Examples of legal identifiers: age, $salary, _value, __1_value.
- Examples of illegal identifiers: 123abc, -salary.

Java Variables

Following are the types of variables in Java −

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

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.

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.

Following are some of the important topics that need to be discussed when looking into classes of the Java Language.

Constructors

When discussing about classes, one of the most important sub topic would be 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.
  }
}
//Types in constructors
```

Creating an Object

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.

Example

```
public class Puppy {
  public Puppy(String name) {
    // This constructor has one parameter, name.
    System.out.println("Passed Name is :" + name );
  }

  public static void main(String []args) {
    Puppy myPuppy = new Puppy( "tommy" );
  }
}
```

Accessing Instance Variables and Methods

Instance variables and methods are accessed via created objects. To access an instance variable, following is the fully qualified path −

```
/* First create an object */
ObjectReference = new Constructor();

/* Now call a variable as follows */
ObjectReference.variableName;
```

/* Now you can call a class method as follows */

ObjectReference.MethodName();

Example

This example explains how to access instance variables and methods of a class.

```
public class Puppy {
  int puppyAge;

  public Puppy(String name) {
    // This constructor has one parameter, name.
    System.out.println("Name chosen is :" + name );
  }

  public void setAge( int age ) {
    puppyAge = age;
  }

  public int getAge( ) {
    System.out.println("Puppy's age is :" + puppyAge );
    return puppyAge;
  }

  public static void main(String []args) {
    /* Object creation */
    Puppy myPuppy = new Puppy( "tommy" );

    /* Call class method to set puppy's age */
    myPuppy.setAge( 2 );

    /* Call another class method to get puppy's age */
    myPuppy.getAge( );

    /* You can access instance variable as follows as well */
    System.out.println("Variable Value :" + myPuppy.puppyAge );
  }
}
```

Java Data Types

Primitive data types:

There are eight primitive data types in Java:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Java - Variable Types

This chapter will explain various variable types available in Java Language. There are three kinds of variables in Java −

- Local variables
- Instance variables
- Class/Static variables

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- 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 can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

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 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.
- 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*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

Access Control 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).

Let's understand the access modifiers in Java by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Java Strings

A String variable contains a collection of characters surrounded by double quotes.

String city=”Hyderabad”;

Methods: length(), toUpperCase(), toLowerCase(), indexOf() , concat()

The + operator can be used between strings to combine them.

```
String firstName = "John";

String lastName = "Doe";

System.out.println(firstName + " " + lastName);

System.out.println(firstName.concat(lastName));
```

Strings are mutable.

Java Arrays

Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable −

Syntax

```
dataType[] arrayRefVar;   // preferred way.

or

dataType arrayRefVar[];  // works but not preferred way.
```

Creating Arrays

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.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below −

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

Alternatively you can create arrays as follows −

dataType[] arrayRefVar = {value0, value1, ..., valuek}; // check out Arrays class
```

JAVA OBJECT ORIENTED CONCEPTS

Key Topics:

- Inheritance
- Overriding
- Polymorphism
- Abstraction
- Encapsulation
- Interfaces

Inheritance:

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

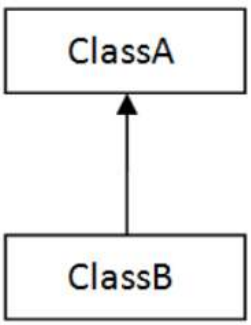
The syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

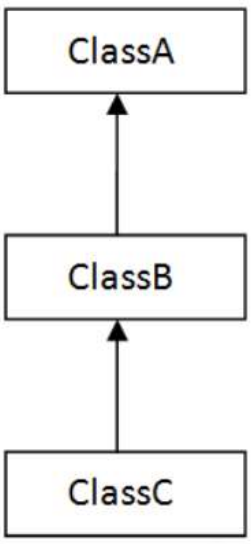
Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

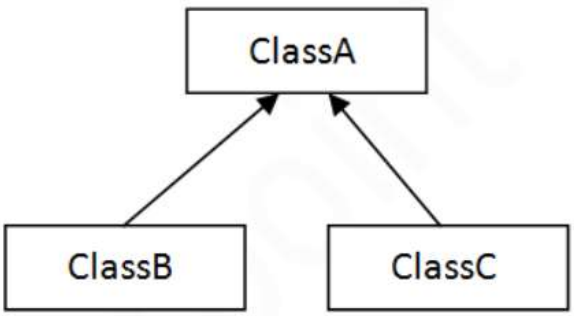
In java programming, multiple and hybrid inheritance is supported through interface only.



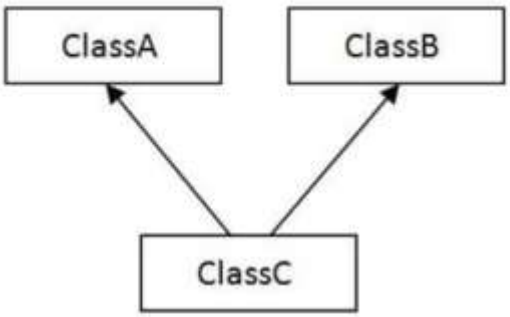
1) Single



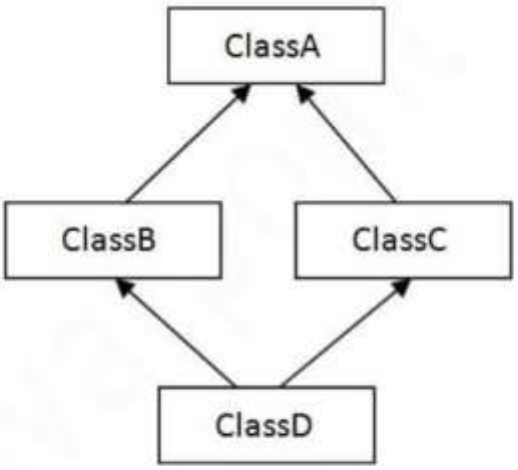
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

Java - Overriding

The benefit of overriding is: ability to define a behavior that's specific to the subclass type, which means a subclass can implement a parent class method based on its requirement.

In object-oriented terms, overriding means to override the functionality of an existing method.

Rules for Method Overriding

1. The argument list and return type should be exactly the same as that of the overridden method.
2. The access level cannot be more restrictive than the overridden method's access level. For example: If the superclass method is declared public then the overridding method in the sub class cannot be either private or protected.
3. A method declared final cannot be overridden.
4. A method declared static cannot be overridden but can be re-declared.
5. If a method cannot be inherited, then it cannot be overridden.
6. Constructors cannot be overridden.

Java - Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Polymorphism in Java has two types: Compile time polymorphism (static binding) and Runtime polymorphism (dynamic binding). Method overloading is an example of static polymorphism, while method overriding is an example of dynamic polymorphism.

Java Abstract Classes and Methods

Data **abstraction** is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either **abstract classes** or interfaces.

The `abstract` keyword is a non-access modifier, used for classes and methods:

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

To access the abstract class, it must be inherited from another class.

Java Interface

Another way to achieve abstraction in Java, is with interfaces.

An `interface` is a completely "**abstract class**" that is used to group related methods with empty bodies:

```
// interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
```

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.

3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Other topics:

- Super keyword
- Final Keyword
- Jre vs jdk vs jvm (https://www.javatpoint.com/difference-between-jdk-jre-and-jvm)
- StringBuffer
- StringBuilder
- Java I/O (https://www.javatpoint.com/java-io)

Java Advanced:

Collections (https://www.javatpoint.com/collections-in-java)

Multithreading (https://www.javatpoint.com/multithreading-in-java)