Overview of Java

Java is a high-level programming language originally developed by Sun Microsystems under the guidance of James Gosling and there team, and released in 1995.

OOPs Concepts in Java

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

1) Object: Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

2) Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

3) Encapsulation: Binding (or wrapping) code and data together into a single unit is known as encapsulation. A java class is the example of encapsulation.

4) Abstraction: Hiding internal details and showing functionality is known as abstraction.

5) Inheritance: The process by which one class acquires the properties and functionalities of another class is called inheritance. It provides code reusability.

6) Polymorphism: When one task is performed by different ways i.e. known as polymorphism. In java, we use method overloading and method overriding to achieve polymorphism.

Features of Java

1) Simple: The Java language is easy to learn and its coding style is easy to read and write.

2) Object Oriented: It has all OOP features such as abstraction, encapsulation, inheritance and polymorphism.

3) Portable: Java programs can execute in any environment (Linux, Window, Mac etc.)

4) Platform Independent: Java is guaranteed to be Write Once, Run Anywhere language.

5) Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems.

6) Robust: Its capability to handle Run-time Error, automatic garbage collection, the lack of pointer concept, Exception Handling etc. makes java robust.

7) Architectural Neutral: To enable a Java application to execute anywhere on the network, the compiler generates an architecture-neutral object file format.

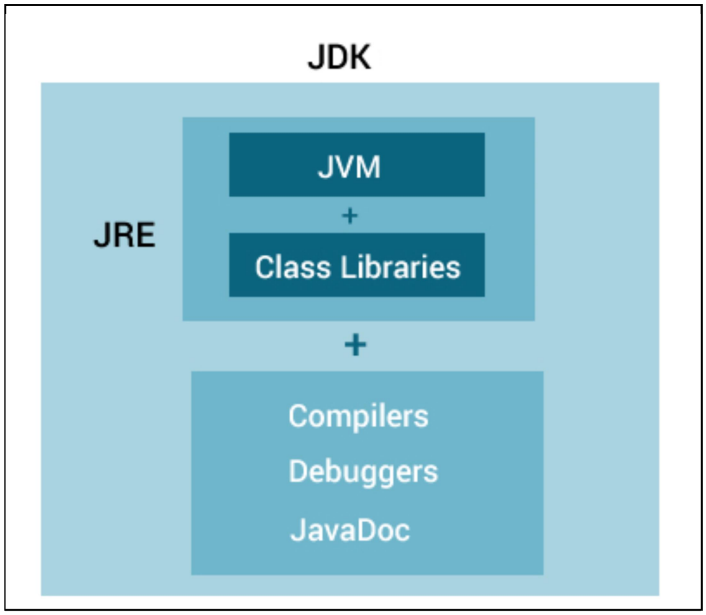
8) Dynamic: It supports Dynamic memory allocation due to this memory wastage is reduce and improve performance of the application.

9) Interpreted: The Java compiler generates byte-codes, rather than native machine code.

10) High Performance: Java enables high performance with the use of just-in-time compiler.

11) Multithreaded: It utilizes same memory and other resources to execute multiple threads at the same time.

12) Distributed: It has networking facilities, so it can be transmit, run over internet.



JDK (Java Development Kit): JDK is a container of tools which are needed to develop java programs.

JRE (Java Runtime Environment): JRE is an implementation of the JVM which actually executes Java programs.

JVM (Java Virtual Machine): JVM is an abstract machine that enables your computer to run a Java program

Data Types in Java

Data types represent the different values to be stored in the variable. In java, there are two types of data types:

1. **Primitive data types:** Primitive data types are predefined by the language and named by a keyword.
2. **Non-primitive data types:** These data types are those which are developed by programmers by making use of appropriate features of the language.

Primitive Data Types

Sr. No.	Data Type	Size	Contains
1.	byte	1 byte	Byte Length Integer
2.	short	2 byte	Short Integer
3.	int	4 byte	Integer
4.	long	8 byte	Long Integer
5.	float	4 byte	Single Precision Floating Point
6.	double	8 byte	Double Precision Floating Point
7.	char	2 byte	A Single Character(Unicode Character)
8.	boolean	1 bit	A Boolean Value(True or False)

Control Statements in Java

The control statements are used to control the flow of execution of the program. Java contains the following types of control statements:

1) Branching Statements (Selection Statements)

Selection statements allow you to control the flow of program execution on the basis of the outcome of an expression or state of a variable known during runtime.

Selection statements can be divided into the following categories:

- **The if statements:** It executes the *if block* if condition is true.

  Syntax:

```
if(condition)
{
//code to be executed
 }
```

- **The if-else statements:** It executes *if block* if condition is true otherwise else block is executed.

  Syntax:

```
if(condition)
{
//code if condition is true
}
else
```

```
{
//code if condition is false
}
```

- **The if-else-if statements (nested if-else statements):** It executes one condition from multiple statements.

Syntax:

```
if(condition1)
{
//code to be executed if condition1 is true
}
else if(condition2)
{
//code to be executed if condition2 is true
}
else if(condition3)
{
//code to be executed if condition3 is true
}
...
else
{
//code to be executed if all the conditions are false
}
```

- **The switch statements:** It is used when we have number of options (or choices) and we may need to perform a different task for each choice.

Syntax:

```
switch(expression)
{
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......
default:
 code to be executed if all cases are not matched;
}
```

2) Looping Statements (Iteration Statements)

Iteration statements execute the same set of instructions until a termination condition is met.

Java provides the following loop for iteration statements:

- **The while loop:** while loop is used to execute statement(s) until a condition holds true.

  Syntax:

```
while(condition)
{
Statements(s);
}
```

- **The for loop:** for loop is a statement which allows code to be repeatedly executed. For loop contains 3 parts Initialization, Condition and Increment or Decrements

  Syntax:

```
for( initialization; condition; increment )
{
 statement(s);
}
```

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

  Syntax:

```
do
{
Statement(s);

increment/decrement (++ or --)
}while();
```

- **The for each loop:** It is mainly used to traverse array or collection elements.

  Syntax:

```
for (type var : array)
{
   statements using var;
}
```

3) Jump Statements

Jump statements are used to unconditionally transfer the program control to another part of the program.

Java provides the following jump statements:

- **break statement:** This statement is used to jump out of a loop.

  Syntax:

```
if(condition)
{
  break;
}
```

- **continue statement:** This statement is used only within looping statements.

Syntax :

```
if(condition)
{
  continue;
}
```

- **return statement:** The return statement is used to explicitly return from a method.

Syntax :

```
if(condition)
{
  return;
}
```

Operators in java

Operators are special symbols that are used to perform mathematical or logical operations on operands (variables or values).

Different types of Operators in java

Operator Type	Category	Precedence
Unary	Postfix	*expr*++ *expr*--
	Prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	Additive	+ -
Shift	Shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	Equality	== !=
Bitwise	bitwise AND	&

	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	Ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

1. **Unary Operators:** Unary operators need only one operand. They are used to increment, decrement or negate a value.
2. **Arithmetic Operators:** They are used to perform simple arithmetic operations on primitive data types.
3. **Shift Operators:** These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively.
4. **Relational Operators:** Relational Operators are used to determine the comparison between two or more objects.
5. **Bitwise Operators:** These operators are used to perform manipulation of individual bits of a number.
6. **Logical Operators:** These operators are used to perform "logical AND" and "logical OR" operation
   - **Logical AND (&&):** returns true when both conditions are true.
   - **Logical OR (||):** returns true if at least one condition is true.
7. **Ternary Operator:** Ternary operator is a shorthand version of if-else statement. It has three operands and hence the name ternary.
   General format is-
   condition ? if true : if false
8. **Assignment Operator:** Assignment operator is used to assign value to the variables, assign memory to object.
   General format is-
   variable = value;

Array

An array is a collection of similar data types that have contiguous memory location.

Advantage of Java Array

- Code Optimization: It makes the code optimized, we can retrieve or sort the data easily.

- Random access: We can get any data located at any index position.

Disadvantage of Java Array

- Size Limit: We can store only fixed size of elements in the array.

Types of Array in java

There are two types of array:

Single Dimensional Array: A one-dimensional array is a linear list of elements of the same type.

Syntax :

```
datatype[ ] identifier;
or
datatype identifier[ ];
```

Example of single dimensional java array

```
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Output: 10
20
70
40
50

Multidimensional Array: MultiDimensional Array is used to store the values in the rows as well as in columns.

Syntax:

```
datatype[ ][ ] identifier;
or
datatype identifier[ ][ ];
```

Example of Multidimensional java array

```
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
```

```
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}

}}
```

Output: 1 2 3
2 4 5
4 4 5

Java Reference Variable

Reference variables are used to refer to an object. They are declared with a specific type which cannot be changed.

Types of reference variables

- Static Variable
- Instance Variable
- Method Parameter
- Local Variable

Method overloading

If two or more method in a class has same name but different parameters, it is known as method overloading. Overloading always occur in the same class (unlike method overriding). Method overloading is one of the ways through which java supports polymorphism.

Advantage of method overloading

Method overloading increases the readability of the program.

Different ways of Method overloading

There are three different ways of method overloading

- Method overloading by changing data type of Arguments
- Method overloading by changing no. of argument.
- Method overloading by changing sequence of data type of arguments

Example:

```
class Addition
{
void sum(int a, int b)
```

```
{
System.out.println(a+b);
}
void sum(int a, int b, int c)
{
System.out.println(a+b+c);
}
public static void main(String args[])
{
Addition obj=new Addition();
obj.sum(10, 20);
obj.sum(10, 20, 30);
}
}
```

Output: 30
60

Method Overriding

Whenever same method name is existing in both base class and derived class with same types of parameters or same order of parameters is known as method Overriding.

Note: Without Inheritance method overriding is not possible.

Advantage of Java Method Overriding

- Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method Overriding is used for Runtime Polymorphism

Example:

```
class Walking
{
void walk()
{
System.out.println("Man walking fastly");
}
}
class Man extends walking
{
void walk()
{
System.out.println("Man walking slowly");
}
```

```
}
class OverridingDemo
{
public static void main(String args[])
{
Man obj = new Man();
obj.walk();
}
        }
```

Output: Man walking slowly

Constructor in Java

Constructor in java is a special type of method that is used to initialize the object.
Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor

There are basically two rules defined for the constructor.

- Constructor name must be same as its class name
- Constructor must have no explicit return type

Types of java constructors

There are two types of constructors:

1) **Default Constructor:** A constructor that have no parameter is known as default constructor.

   Syntax:

   ```
   <class_name>(){}
   ```

   Example:

   ```
   class Bike
   {
   Bike()
   {
   System.out.println("Bike is created");
   }
   public static void main(String args[])
   {
   Bike b=new Bike();
   }
   }
   ```

   Output: Bike is created

2) **Parameterized constructor:** A constructor that has parameters is known as parameterized constructor.

Syntax:

<class_name>(parameters){}

Example:

```
class Student
{
   int id;
   String name;
   Student(int i,String n)
  {
   id = i;
   name = n;
   }
   void display()
{
System.out.println(id+" "+name);
}
   public static void main(String args[]){
   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan");
   s1.display();
   s2.display();
  }
}
```

Output: 111 Karan
222 Aryan

Constructor Overloading

Constructor overloading is a concept of having more than one constructor with different parameters list, in such a way so that each constructor performs a different task.

Example:

```
class Student{
   int id;
   String name;
   int age;
   Student(int i,String n)
  {
   id = i;
   name = n;
   }
```

```
    Student(int i,String n,int a)
    {
    id = i;
    name = n;
    age=a;
    }
    void display()
    {
    System.out.println(id+" "+name+" "+age);
    }
    public static void main(String args[])
    {
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(222,"Aryan",25);
    s1.display();
    s2.display();
    }
}
```

Output: 111 Karan 0
222 Aryan 25

Inheritance in Java

Inheritance in java is a mechanism in which one class acquires the properties(data members) and functionalities(methods) of another class.

Syntax :

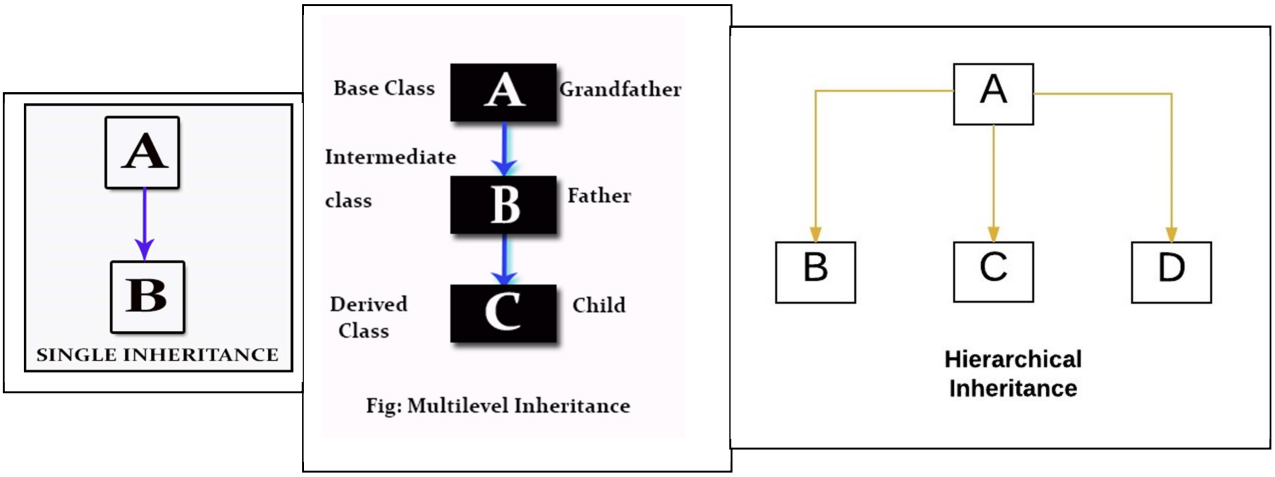
```
class derived-class extends base-class
{
  //methods and fields
}
```

Types of inheritance in java

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

1) **Single Inheritance:** In single inheritance, subclasses inherit the features of one superclass.

2) **Multilevel Inheritance:** In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class.
3) **Hierarchical Inheritance:** In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.



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

4) **Multiple Inheritance (Through Interfaces):** In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Please note that Java does not support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces.
5) **Hybrid Inheritance (Through Interfaces):** It is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.