1. **JVM?**

Java Virtual Machine is a virtual machine that enables the computer to run the Java program. JVM acts like a run-time engine which calls the main method present in the Java code. JVM is the specification which must be implemented in the computer system. The Java code is compiled by JVM to be a Bytecode which is machine independent and close to the native code.

1. **What are Variables?**

Variables are used to store the data and it is declared by specifying the data type.

**Different Data Types:**

|  |  |
| --- | --- |
| **Primitive Data Type** | **Non Primitive Data Type** |
| It is pre-defined in java | It is created by the programmer and not defined by Java except String |
| It has always a value | It can be a null |
| Byte  Short  Char  Int  Float  Double  long | String  Arrays  Classes |

1. **Local vs Global variable**

|  |  |
| --- | --- |
| **Local Variable** | **Global Variable** |
| A variable which is declared inside the function or block is known as local variable | A variable which is declared outside the function or block is known as global variable |
| It can not be accessed outside the function | It can be accessed outside the function |
| Scope of this variable is available within the function only | Scope of this variable is available throughout the program |

1. **Type casting?**

The process of converting the value from one data type to another data type is called Type casting. We have two different kind of type casting.

1. **Implicit Casting:**

In the implicit type conversion it automatically convert one data type to another.

Eg:

int myNum=5;

double result= myNum; 🡪 o/p: 5.0

1. **Explicit Casting:**

In this type, user manually convert the data from one type to another.

Eg:

double result = 5.67;

int myNum = (int)result; 🡪o/p: 5

1. **Ternary Operator:**

It’s a shorthand for an if else codition. It contains 3 operands. Based on the condition it will assign the value.

Syntax:

Variable = (condition) ? exp true : exp false;

**Eg:**

Int time =18;

String Result = (time<12) ? “Good mrng” : “Good evng”;

o/p: Good evng

1. **Loops:**

**While Loop:**

It will execute a block of code ‘n’ number of times until the condition is getting failed.

Syntax:

while(condition){

-----

-----

}

**Do while Loop:**

It will execute the code once and then check the condition.

Syntax:

do {

----

----

}while(condition);

**For loop:**

If we know the number of iteration we need to execute the code then we can use for loop.

Syntax:

for(declaration; condition; inc/dec){

----

----

}

**For-Each Loop:**

It will execute the block of code by iterating the elements of list, arrays or collections

Syntax:

for(DataType variableName : ArrayName){

-----

-----

}

Eg:

String[] cars = {“volvo”, “BMW”, “Ford”, “Benz”}

for (String car : cars){

sysout(car);

}

1. **Break vs Continue:**

|  |  |
| --- | --- |
| **Break** | **Continue** |
| Allows loop termination when some condition is met and the control is moved to the next statement | Allows skipping some part of loop when some conditions is met and continue to the next iteration in the loop |
| for(int i=0;i<6;i++){  if(i==4){  break;  }  Sysout(i)  } | for(int i=0;i<6;i++){  if(i==4){  continue;  }  Sysout(i)  } |
| Output: 0 1 2 3 | Output: 0 1 2 3 5 |

1. **Different Access Specifier:**

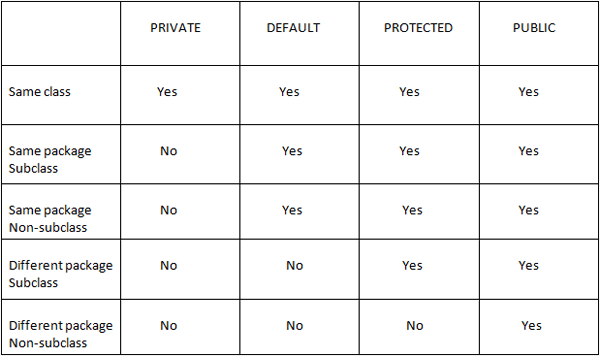
Access Specifiers are keywords which are used to define the scope of the class, methods or variables. There are three different types of access specifier.

**Public**: A class, method or variable which are declared as public can be access by any class or methods.

**Private**: A class, methods or variable which are declared as private can be accessed within the class only.

**Protected**: Can be accessed by the class of the same package or the different package subclass.

**Default**: Can be accessed by the class of the same package.



1. **OOPS – Object Oriented Programming System:**

OOPS is a methodology to design a programming using classes and Objects.

* Object
* Class
* Encapsulation
* Polymorphism
* Inheritance
* Abstraction

1. **Object:**

Its an entity that has state and behaviour. In the program Object can be defined as an instance of the class. In class state define by the variables and behaviour define by the methods.

1. **Class:**

A class is a blueprint from which we can create an individual object

1. **Inheritance:**

It’s a mechanism by which one class is allowed to acquire all the properties and behaviours of another class. In Java we have 3 types of inheritance.

1. **Single Inheritance:**

Here we have one subclass and one superclass. The subclass will inherit the properties and behaviour of the superclass.

Eg:

class car{

public void run() {

System.out.println("Car is running");

}

}

class BMW extends car{

public void company() {

System.out.println("I am a BMW car");

}

}

public class singleLevelInheritance {

public static void main (String[] args) {

BMW mybmw = new BMW();

mybmw.run();

mybmw.company();

}

}

1. **Multilevel Inheritance:**

Here we have one grandparent class, one parent and child class. The child class will inherit the parent class and parent class inherit the grandparent class. So, if we create object for the child class, we can able to access the properties and behaviour of the both parent and grandparent class.

Eg:

class car{

public void run() {

System.out.println("I am running");

}

}

class flyingCar extends car{

public void fly() {

System.out.println("I am flying");

}

}

class BMW extends flyingCar{

public void company() {

System.out.println("I am a BMW car");

}

}

public class MultiLevelInheritance {

public static void main (String[] args) {

BMW mybmw = new BMW();

mybmw.company();

mybmw.run();

mybmw.fly();

}

}

1. **Hierarchical Inheritance:**

In which a parent class can have two or more child class.

Eg:

class car{

public void run() {

System.out.println("I am running");

}

}

class BMW extends car{

public void company() {

System.out.println("I am a BMW car");

}

}

class Ford extends car{

public void company() {

System.out.println("I am a Ford car");

}

}

public class HierarchicalInheritance {

public static void main (String[] args) {

BMW mybmw = new BMW();

mybmw.company();

mybmw.run();

Ford myford = new Ford();

myford.company();

mybmw.run();

}

}

1. **Polymorphism.**

The word polymorphism means many forms. We can define polymorphism is the ability of a message to displayed in more than one forms. We are two kinds of polymorphism.

* Compile time polymorphism
* Run time polymorphism

1. **Compile time polymorphism – Method Overloading**

If a class had two or more method with the same name but with different parameter or data type then it called Method Overloading.

Eg.:

class Calculation {

public void add(int a, int b) {

System.out.println("Result : "+ (a+b));

}

public void add(int a, int b, int c) {

System.out.println("Result : "+ (a+b+c));

}

public void add(double a, double b) {

System.out.println("Result : "+ (a+b));

}

}

public class Compile\_MethodOverloading {

public static void main(String[] args) {

Calculation calc= new Calculation();

calc.add(2,4);

calc.add(2,4,5);

calc.add(5.2, 5.2);

}

}

1. **Run Time Polymorphism – Method Overriding**

If the child class has the same method name as declared in the parent class, then it is known as Method Overriding.

In which, during runtime only java knows which class method need to call based on the object.

Eg.:

class Bank{

public void interest() {

System.out.println("2% Interest");

}

}

class IndianBank extends Bank{

public void interest() {

System.out.println("5% Interest");

}

}

public class Runtime\_MethodOverriding {

public static void main (String[] args) {

IndianBank mybank = new IndianBank();

mybank.interest();

}

}

1. **Encapsulation:**

It is the process of wrapping the code and data together in a single unit. It is used to protect the data from the outer class. And it can be accessed or modified using getters and setters methods.

Eg.:

class BankLoan{

private String name;

private int age;

private double amount;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public double getAmount() {

return amount;

}

public void setAmount(double amount) {

this.amount = amount;

}

}

public class Encapsulation {

public static void main(String[] args) {

BankLoan loan = new BankLoan();

loan.setName("Arun");

loan.setAge(20);

loan.setAmount(500);

System.out.println(loan.getName()+" age is "+loan.getAge()+" and he bought a loan of $"+loan.getAmount());

}

}

1. **Abstraction:**

Abstraction refers to providing only the essential information about the data to the outer class and hiding the background implementation.

Or

Abstraction means hiding the implementation details and showing only the functionality to the user. It focuses on what the object does instead of how it does it.

In java we can achieve abstraction using abstract class and interface.

1. **Abstract class:**

Class which is declared with the abstract keyword is called abstract class. It can have abstract and non-abstract method. Abstract method is a method without implementation

Rules:

* Creation of object for the abstract class is not allowed and the Abstract class needs to be extended
* Abstract class must have at-least one abstract method

Eg:

abstract class shape{

abstract public void displayShape();

public void display() {

System.out.println("I am from Shape class");

}

}

class circle extends shape{

public void displayShape() {

System.out.println("I am circle");

}

}

public class abstractClass {

public static void main(String[] args) {

circle c=new circle();

c.display ();

c.displayShape();

}

}

1. **Interface:**

Interface is used to achieve 100% abstraction. Using ‘Interface’ keyword we can create interface class

Rules:

* In interface only abstract methods are allowed.
* All variables declared in interface should be constant i.e. public static final
* We can’t create object for the interface and we need to implemet the interface class
* An interface can also extend another interface class

Eg:

interface UniversityDetails{

public static final String University ="SASTRA";

public void NoOfDepartments();

public void NoOfStudents();

}

class Sastra **implements** UniversityDetails{

public void NoOfDepartments() {

System.out.println("We had 12 department");

}

public void NoOfStudents() {

System.out.println("We had 500+ students");

}

}

public class interfaceClass{

public static void main(String[] args) {

Sastra univ = new Sastra();

univ.NoOfDepartments();

univ.NoOfStudents();

}

}

1. **super, this, static and Final Keywords**
2. **super Keyword:**

Super Keyword is used to access the methods or variables of the immediate super class (parent class).

Eg:

class parentClass{

String classname="Parent Class variable";

parentClass(){

System.***out***.println("I am a Parent Class constructor");

}

public void name() {

System.***out***.println("I am from Parent Class");

}

}

class childClass extends parentClass{

String classname="Child Class variable";

childClass(){

super(); //if we didn't mention then also parent class constructor will be called

System.***out***.println("I am a Child Class constructor");

}

public void name() {

System.***out***.println("I am from child Class");

}

public void printNames() {

//Super Keyword - Variable

System.***out***.println(super.classname);

System.***out***.println(classname);

//Super Keyword - Method

super.name();

name();

}

}

public class SuperKeyword {

public static void main (String[] args) {

childClass c1= new childClass();

c1.printNames();

}

}

1. **this Keyword:**

this keyword is used to refer the current class instance variable or methods or Constructor.

class Student{

int rollNo;

String Name;

Student(int rollNo, String Name){

this.rollNo=rollNo;

this.Name=Name;

}

public void display() {

System.***out***.println("Name: "+Name+" Roll No: "+rollNo);

}

}

public class ThisKeyword {

public static void main(String[] args) {

Student s1= new Student(123,"Ragul");

Student s2= new Student(456,"Arun");

s1.display();

s2.display();

}

}

1. **static Keyword**

Static Keyword is used to create class level variables and methods.

Static member of a class belongs to class itself instead of object.

* static – Belongs to the class
* non static – Belongs to the class object

Eg:

* package keywords;
* class CounterClass{
* int i=0;
* static int *j*=0;
* public CounterClass(){
* i++;
* *j*++;
* System.***out***.println("i :"+i +" j :"+*j*);
* }
* }
* public class StaticKeyword {
* public static void main(String[] args) {
* CounterClass C1 = new CounterClass();
* CounterClass C2 = new CounterClass();
* CounterClass C3 = new CounterClass();
* }
* }

Output:

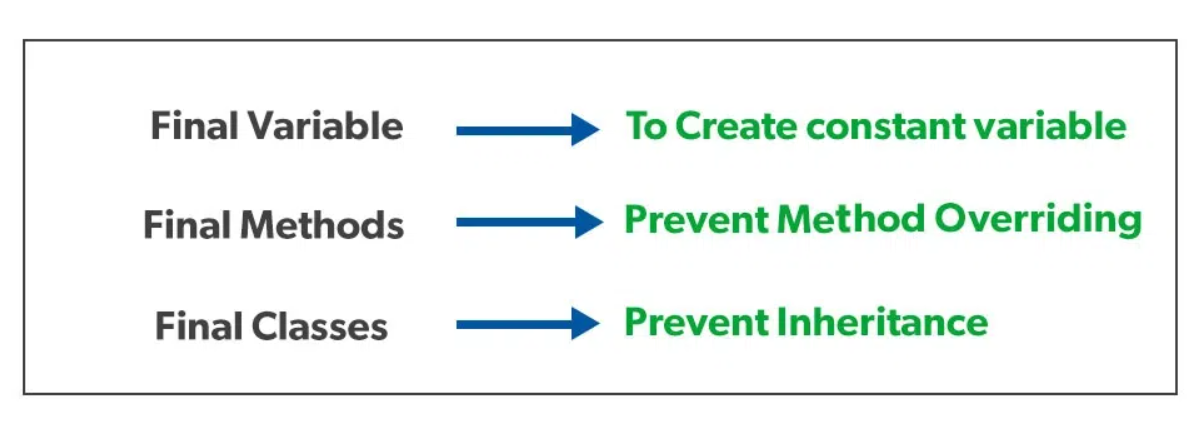
i :1 j :1

i :1 j :2

i :1 j :3

1. **Final Keyword:**

The final keyword is a non-access modifier applicable only to a variable, a method, or a class. It is used to restrict a user in Java.



1. **Exceptional Handling:**

It’s a mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

Using try, catch, finally block we can handle the exception.

Some common Java Exception:

1. NullPointerException
2. ArithmeticException
3. ArrayIndexOutOfBoundsException
4. FileNotFoundException
5. NoSuchFieldException

Some common Selenium Exception:

1. NoSuchElementException.
2. ElementNotVisibleException.
3. NoSuchFrameException.
4. NoAlertPresentException.
5. NoSuchWindowException.
6. SessionNotFoundException.
7. StaleElementReferenceException.
8. InvalidSelectorException.
9. List vs Set vs Map in Java

| **List** | **Set** | **Map** |
| --- | --- | --- |
| The list interface allows duplicate elements | The set does not allow duplicate elements. | The map does not allow duplicate elements |
| The list maintains insertion order. | The set does not maintain any insertion order. | The map also does not maintain any insertion order. |
| We can add any number of null values. | But in the set almost only one null value. | The map allows a single null key at most and any number of null values. |
| The list implementation classes are Array List and LinkedList. | Set implementation classes are HashSet, LinkedHashSet, and TreeSet. | Map implementation classes are HashMap, HashTable, TreeMap, ConcurrentHashMap, and LinkedHashMap. |