

Contents

OOPS Concepts	2
Types of Polymorphism.....	2
Difference between Method overloading and Method overriding	3
Constructor	3
Types of Constructors	3
Static Keyword – non access modifier	4
Final– Cannot change value	5
Difference between Final & static	5
Access Modifiers or Access Specifiers.....	5
Inheritance	6
Types of Inheritance	6
Data Abstraction	7
Abstract class in Java.....	7
Points to Remember	7
Abstract Method in Java	7
Interface	8
Which one is pure abstract? - Interface.....	9
Which one gives 100% abstraction? - Interface	9
Difference between Abstract and interface	9
Exception Handling	10
What is Exception Handling?	10
Types of Java Exceptions.....	10
Difference between Checked and Unchecked Exceptions	11
1) Checked Exception.....	11
2) Unchecked Exception.....	11
3) Error	11
Java Exception Keywords	11
Java Exception Handling Example	12
Syntax of Java try-catch	12
Syntax of Java try-finally	12
Difference between final, finally, finalize	13
Garbage Collection.....	13
Super, This.....	14

Collections.....	14
Basic programs.....	14
Interview questions	15
Is java 100% object oriented?	15
Instance & Local Variable.....	15
difference between equals() method and equality operator (==).....	15
Enum	16
Stack & Heap.....	16
Read a file.....	16
Write a file.....	17
String.....	17
Difference between String Buffer & builder	17
Collections.....	17
String.....	17
Static & volatile	17
Singleton Class	17
How to design a singleton class?	18

Data Members = int a;String b;

Member function (Methods)= void run() {}

OOPS Concepts

Class -Binds both data member & member function

Objects- instance of the class, Property of the class

Inheritance- sub class inherits the property of parent class. Code reusability

Polymorphism – Takes more than one form

Data Abstraction- without giving background info

Data Encapsulation- Wrapping up of data into single unit

Types of Polymorphism

Compile Time	Run Time
Static Binding	Dynamic Binding
Eg: Method Overloading	Eg: Method Overriding
Early Binding	Late Binding

Difference between Method overloading and Method overriding

Method Overloading	Method Overriding
Parameter must be different (No of parameter)	Parameter must be same
Mostly used in same class	Mostly used between different classes
<pre> public class MethodOverloading { public static void main(String[] args) { // TODO Auto-generated method stub int x=add(2,3); int y=add(2,3,5); System.out.println(x); System.out.println(y); } static int add(int a,int b) { int c=a+b; return c; } static int add(int a,int b,int c) { c=a+b+c; return c; } } </pre>	<pre> class parentclass{ int add(int a,int b) { return a+b; } } public class MethodOverriding { int add(int a,int b) { return a+b; } public static void main(String[] args) { // TODO Auto-generated method stub parentclass parent=new parentclass(); int x=parent.add(5, 5); MethodOverriding child=new MethodOverriding(); int y=child.add(10, 5); System.out.println(x); System.out.println(y); } } </pre>

Constructor

Constructor is used to create memory.

It is initialized when the object of relative class is called.

Constructor name must be the same as its class name

A Constructor must have no explicit return type (primitive data type, void, String)

A Java constructor cannot be abstract, static, final, and synchronized

Types of Constructors

Default constructor

Parameterized constructor

Default	Parameterized
No argument/Parameters	It will have arguments/Parameters

<pre> public class DefaultConstructor { static int x; DefaultConstructor() { System.out.println("Calling Default Constructor"); x=10; } public static void main(String[] args) { // TODO Auto-generated method stub DefaultConstructor DefCon=new DefaultConstructor(); int y=10+x; System.out.println(y); } } </pre>	<pre> public class ParameterizedConstructor { static int a,b; ParameterizedConstructor(int x,int y) { a=x; b=y; System.out.println("ParameterizedConstructor"); } public static void main(String[] args) { // TODO Auto-generated method stub ParameterizedConstructor PC=new ParameterizedConstructor(4,5); System.out.println(a + " " + b); } } </pre>
---	--

Static Keyword – non access modifier

Variable ,Method, Block ,Nested Classes

Variables	Method	Block
single copy of variable is created and shared among all objects at class level	A static method can access static data member/Member function and can change the value of it.	initialize the static data member
Similar to Global Variable		executed before the main method at the time of classloading

<pre> 3 public class Static_met { 4 static int x = 1; 5 // Static variable 6 public static void main(String[] args) { 7 8 System.out.println("Printing static " +x) 9 x++; 10 System.out.println("Incrementing static v 11 add(); 12 } 13 //static method 14 static void add() { 15 x = x + 1; 16 System.out.println("Inside static method" 17 System.out.println(x); 18 } </pre> <p>Console</p> <pre> <terminated> Static_met [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (25-Sep-202 Printing static 1 Incrementing static value 2 Inside static method 3 </pre>	<pre> 1 package staticvarmet; 2 3 public class StaticBlock { 4 static { 5 System.out.println("static block is inv 6 } 7 8 public static void main(String[] args) { 9 // TODO Auto-generated method stub 10 } 11 } 12 13 } 14 </pre> <p>Console</p> <pre> <terminated> StaticBlock [Java Application] C:\Program Files\Java\jdk static block is invoked </pre>
---	--

Final– Cannot change value

A final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

Variable	Method	Class
Cannot change the value of final variable.	Cannot override	Cannot extend
Constant <pre> 3 public class Final_Var { 4 final int x=10; 5 void chnage() 6 { 7 x=20; 8 } 9 } 10 public static void main(String[] args) { 11 Final_Var var=new Final_Var(); 12 var.chnage(); 13 } 14 </pre>	<pre> 3 class parentclass 4 { 5 final void run() 6 { 7 System.out.println("Parent-Final"); 8 } 9 } 10 public class Final_Method extends parentclass{ 11 void run() 12 { 13 System.out.println("Child-Final"); 14 } 15 } 16 public static void main(String[] args) { 17 Final_Method finvar=new Final_Method(); 18 finvar.run(); 19 } 20 21 </pre>	<pre> 3 final class parentclass {} 4 5 public class Final_Class extends parentclass{ 6 void run() 7 { 8 System.out.println("Child-Final"); 9 } 10 } 11 public static void main(String[] args) { 12 Final_Class clasnam=new Final_Class(); 13 clasnam.run(); 14 } 15 </pre>

Difference between Final & static

Static	Final
Static keyword is applicable to nested static class, variables, methods and block.	Final keyword is applicable to class, methods and variables.
It is not compulsory to initialize the static variable at the time of its declaration.	It is compulsory to initialize the final variable at the time of its declaration.
The static variable can be reinitialized.	The final variable can not be reinitialized.
Static methods can only access the static members of the class, and can only be called by other static methods.	Final methods can not be inherited.
Static class's object can not be created, and it only contains static members only.	A final class can not be inherited by any class.
Static block is used to initialize the static variables.	Final keyword supports no such block.

Access Modifiers or Access Specifiers

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

There are four types of Java access modifiers:

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

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

Inheritance

To use the parent class property to child class

Reusability of code

Extends keyword is used

Types of Inheritance

Single -> Parent -> Child

Multilevel -> Grandfather->Father->Son

Hierarchical ->Grandpa->Aunt->Uncle

Hybrid

Multiple – (Not supported in Java. We use interface) ->Mom->Dad->Child

Single	Multi level	Hierarchical
Parent -> Child	Grandfather->Father->Son	Grandpa->Aunt->Uncle
<pre> 9 // Parent class 10 class Animal { 11 void eat() { 12 System.out.println("eating..."); 13 } 14 } 15 //Child class 16 class Dog extends Animal { 17 void bark() { 18 System.out.println("barking..."); 19 } 20 } 21 22 public class SingleInheritance { 23 24 public static void main(String args[]) { 25 Dog d = new Dog(); 26 d.bark(); 27 d.eat(); 28 } 29 } 30 </pre> <p>Console</p> <pre> <terminated> SingleInheritance [Java Application] C:\Program File barking... eating... </pre>	<pre> 3 class Granpa { 4 void eat() { 5 System.out.println("eating..."); 6 } 7 } 8 9 class father extends Granpa { 10 void bark() { 11 System.out.println("barking..."); 12 } 13 } 14 15 class Son1 extends father { 16 void weep() { 17 System.out.println("weeping..."); 18 } 19 } 20 21 public class MultilevelInheritance { 22 23 public static void main(String args[]) { 24 Son1 d = new Son1(); 25 d.weep(); 26 d.bark(); 27 d.eat(); 28 } 29 } 30 </pre> <p>Console</p> <pre> <terminated> MultiLevelInheritance [Java Application] C:\Program Files weeping... barking... eating... </pre>	<pre> 2 3 class Parents { 4 void eat() { 5 System.out.println("eating...");}} 6 class Daughter extends Parents { 7 void bark() { 8 System.out.println("barking...");}} 9 class Son extends Parents { 10 void meow() { 11 System.out.println("meowing...");}} 12 public class HierarchicalInheritance { 13 public static void main(String args[]) { 14 Son c = new Son(); 15 c.meow(); 16 c.eat(); 17 18 Daughter d= new Daughter(); 19 d.bark(); 20 d.eat();//C.T.Error 21 } 22 } </pre> <p>Console</p> <pre> <terminated> HierarchicalInheritance [Java Application] C:\Progra meowing... eating... barking... eating... </pre>

Data Abstraction

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Abstract class in Java

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods/Concrete methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Abstract Method in Java

- A method which is declared as abstract and does not have implementation is known as an abstract method.
- abstract void printStatus();//no method body and abstract

```

3 //Abstract class
4 abstract class Animal {
5 // Abstract method (does not have a body)
6 public abstract void animalSound();
7 public abstract void animalSound2();
8 // Regular method
9 public void sleep()
10 {
11 System.out.println("Zzz");
12 }}
13 //Subclass (inherit from Animal)
14 class Pig extends Animal {
15 public void animalSound() {
16 // The body of animalSound() is provided here
17 System.out.println("The pig says: wee wee");
18 }
19 @Override
20 public void animalSound2() {}
21 public class Abstrat_demo {
22 public static void main(String[] args) {
23 Pig myPig = new Pig(); // Create a Pig object
24 myPig.animalSound();
25 myPig.sleep();
26 }

```

Console

```

<terminated> Abstrat_demo [Java Application] C:\Program Files\Java\jdk-1
The pig says: wee wee
Zzz

```

Interface

- An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.
- The interface in Java is a *mechanism to achieve* abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.
- In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.
- It cannot be instantiated just like the abstract class.
- Since Java 8, we can have default and static methods in an interface.
- Since Java 9, we can have private methods in an interface.


```

1 package abstract_Interface;
2 //Interface
3 interface Animal1 {
4     public void animalSound(); // interface method (does not have a body)
5     public void sleep(); // interface method (does not have a body)
6 }
7 // Pig "implements" the Animal interface
8 class Pig1 implements Animal1 {
9     public void animalSound() {
10         // The body of animalSound() is provided here
11         System.out.println("The pig says: wee wee"); }
12     public void sleep() {
13         // The body of sleep() is provided here
14         System.out.println("Zzz");
15     }}
16 public class Interface_Demo {
17     public static void main(String[] args) {
18         Pig1 myPig = new Pig1(); // Create a Pig object
19         myPig.animalSound();
20         myPig.sleep();
21     }
22 }
23
24

```

Console

```

terminated> Interface_Demo [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (25-Sep-202
he pig says: wee wee
zz

```

Which one is pure abstract? - Interface

Which one gives 100% abstraction? - Interface

Difference between Abstract and interface

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: <pre>public abstract class Shape{ public abstract void draw(); }</pre>	Example: <pre>public interface Drawable{ void draw(); }</pre>

Exception Handling

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

- Checked Exception
- Unchecked Exception
- Error

Difference between Checked and Unchecked Exceptions

1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords

Java provides five keywords that are used to handle the exception. The following table describes each.

Keyword	Description
Try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
Catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

Finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
Throw	The "throw" keyword is used to throw an exception.
Throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Java Exception Handling Example

```

3 public class ArithmeticExceptionExample {
4     public static void main(String[] args) {
5         int a = 0, b = 10 ;
6         int c = 0;
7         try {
8             c = b/a;
9         } catch (ArithmeticException e) {
10             e.printStackTrace();
11             // System.out.println("We are just printing the stack tr
12         }
13         System.out.println("Value of c :"+ c);
14     }
15 }
16 /*public static void main(String[] args) {
17     int a = 0, b = 10;

```

Console

```

<terminated> ArithmeticExceptionExample (1) [Java Application] C:\Program Files\Java\jdk-17\
java.lang.ArithmeticException: / by zero
    at exceptionHandling.ArithmeticExceptionExample.main(ArithmeticE
Value of c :0

```

Syntax of Java try-catch

```

try{
    //code that may throw an exception
}catch(Exception_class_Name ref){}

```

Syntax of Java try-finally

```

try{
    //code that may throw an exception

```

}finally{

Difference between final, finally, finalize

Final	Finally	finalize
final is the keyword and access modifier which is used to apply restrictions on a class, method or variable.	finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.
Final keyword is used with the classes, methods and variables.	Finally block is always related to the try and catch block in exception handling.	finalize() method is used with the objects.
(1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited.	(1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block	finalize method performs the cleaning activities with respect to the object before its destruction.
Final method is executed only when we call it.	Finally block is executed as soon as the try-catch block is executed. It's execution is not dependant on the exception.	finalize method is executed just before the object is de

Garbage Collection

- Just before destroying an object, Garbage Collector calls *finalize()* method on the object to perform cleanup activities. Once *finalize()* method completes, Garbage Collector destroys that object.
- finalize()* method is present in Object class with following prototype.

- protected void finalize() throws Throwable
- Based on our requirement, we can override *finalize()* method for perform our cleanup activities like closing connection from database.

Super, This

- Super is a reference variable which is used to refer immediate parent class object.

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

This

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.

Collections

List, Set, Queue, Map

String

Basic programs

Odd, Even

Leap Year

Swap numbers

Fibonacci

Factorial

Palindrome

Armstrong

Square root of a number

Matrix Addition

Matrix Multiplication

Transpose of a matrix

Missing number in an array

Find number of words in a string

Temperature to Fahrenheit

Sort (Quick, Insert, bubble)

Prime number

Reverse a string, number

Remove duplicate in an array

Reverse an array

Reverse words of sentences

Interview questions

Is java 100% object oriented?

Instance & Local Variable

Instance variable - accessible by all the methods in the class.

Local variables are those variables present within a block, function, or constructor and can be accessed only inside them

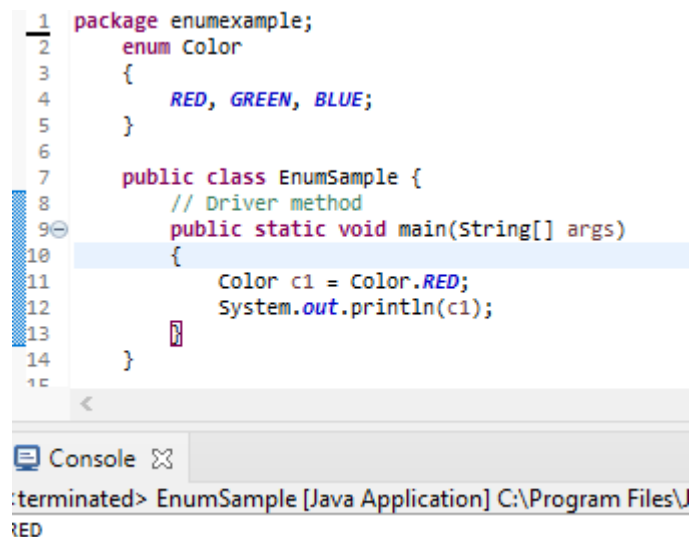
difference between equals() method and equality operator (==)

equals()	==
This is a method defined in the Object class.	It is a binary operator in Java.
This method is used for checking the equality of contents between two objects as per the specified business logic.	This operator is used for comparing addresses (or references), i.e checks if both the objects are pointing to the same memory location.

Enum

Enumerations serve the purpose of representing a group of named constants in a programming language

Enums are used when we know all possible values at **compile time**, such as choices on a menu, rounding modes, command line flags, etc. It is not necessary that the set of constants in an enum type stay **fixed** for all time.



```

1 package enumexample;
2 enum Color
3 {
4     RED, GREEN, BLUE;
5 }
6
7 public class EnumSample {
8     // Driver method
9     public static void main(String[] args)
10    {
11        Color c1 = Color.RED;
12        System.out.println(c1);
13    }
14 }

```

Console

```

:terminated> EnumSample [Java Application] C:\Program Files\J
RED

```

Stack & Heap

- Java Heap Space is used throughout the application, but Stack is only used for the method — or methods — currently running.
- The Heap Space contains all objects are created, but Stack contains any reference to those objects.
- Objects stored in the Heap can be accessed throughout the application. Primitive local variables are only accessed the Stack Memory blocks that contain their methods.
- Memory allocation in the Heap Space is accessed through a complex, young-generation, old-generation system. Stack is accessed through a last-in, first-out (LIFO) memory allocation system.
- Heap Space exists as long as the application runs and is larger than Stack, which is temporary, but faster.

Read a file 10 PM to 12 AM

8:30 AM to 5:00 PM PST

Write a file

String

Difference between String Buffer & builder

StringBuffer		StringBuilder
1)	StringBuffer is <i>synchronized</i> i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is <i>non-synchronized</i> i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is <i>less efficient</i> than StringBuilder.	StringBuilder is <i>more efficient</i> than StringBuffer.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

Collections

String

Static & volatile

static simply means not associated with an instance of the containing class.

volatile simply means that the value may be changed by other threads without warning.

Singleton Class

A Singleton class in Java allows only one instance to be created and provides global access to all other classes through this single object or instance. Similar to the static fields, The instance fields(if any) of a class will occur only for a single time.

How to design a singleton class?

To design a singleton class, we need to do the following things:

- Firstly, declare the constructor of the Singleton class with the private keyword. We declare it as private so that no other classes can instantiate or make objects from it.
- A private static variable of the same class that is the only instance of the class.
- Declare a static factory method with the return type as an object of this singleton class.

```
//Class 1
//Helper class
class Singleton {
    // Static variable reference of single_instance
    // of type Singleton
    private static Singleton single_instance = null;

    // Declaring a variable of type String
    public String s;

    // Constructor
    // Here we will be creating private constructor
    // restricted to this class itself
    private Singleton() {
        s = "Hello I am a string part of Singleton class";
    }
    // Static method
    // Static method to create instance of Singleton class
    public static Singleton getInstance() {
        if (single_instance == null)
            single_instance = new Singleton();
        return single_instance;
    }
}
// Class 2
// Main class
public class SingEg {
    public static void main(String args[]) {
        // Instantiating Singleton class with variable x
        Singleton x = Singleton.getInstance();
        // Instantiating Singleton class with variable y
        Singleton y = Singleton.getInstance();
        // Instantiating Singleton class with variable z
        Singleton z = Singleton.getInstance();
        // Printing the hash code for above variable as
        // declared
        System.out.println("Hashcode of x is " + x.hashCode());
        System.out.println("Hashcode of y is " + y.hashCode());
        System.out.println("Hashcode of z is " + z.hashCode());
        // Condition check
        if (x == y && y == z) {
            System.out.println("Three objects point to the same memory location on the heap i.e, to the same object");
        }
        else {
            System.out.println("Three objects DO NOT point to the same memory location on the heap");
        }
    }
}
```

Hashcode of x is 1072591677

Hashcode of y is 1072591677

Hashcode of z is 1072591677

Three objects point to the same memory location on the heap i.e, to the same object