Interview Preparation

OOPs

★ Simula is considered the first object-oriented programming language.

★ Smalltalk is considered the first truly object-oriented programming language.

OOPs - Object-Oriented programming is a methodology or paradigm to design a program using classes and objects.

It simplifies software development and maintenance by providing some concepts:-

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

Apart from these concepts, there are some other terms which are used in object-oriented design:

- Coupling
- Cohesion
- Association
- Aggregation
- Composition

Object - Any entity that has state and behavior. An Object can be defined as an instance of a class. It can be physical or logical.

for example - A dog is an object because it has states like color, name, breed etc, as

well as behaviours like wagging the tail, barking, eating etc.

Class - Collection of objects is called class. It is a logical entity. A class can also be defined as a blueprint from which you can create an individual object.

object	class
Memory (space)	(space X)

Inheritance -

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance.

It provides code reusability.

It is used to achieve runtime polymorphism.

Polymorphism -

If one task is performed in different ways, it is known as polymorphism. for example - Can be to speak something, a cat speaks meow, dog barks woof, etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. for example - phone call, we don't know the internal processing.

Encapsulation -

Binding (or wrapping) code and data together into a single unit are known as encapsulation. for example - a capsule, it is wrapper with different medicines.

Java Bean is the fully encapsulated class because all the data members are private here.

Coupling -

It refers to the knowledge or information or dependency of another class.

If a class has the details information of another class, there is strong coupling.

Cohesion.

It refers to the level of a component which performs a single well-defined task.

- Java.io package is a highly cohesive package because it has I/O related classes and interface.
- java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

Association. - It represents the relationship between the objects. for example - One Country can have one prime ministers (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many)

Aggregation- It is a way to achieve Association. It represents the relationship where one object contains other objects as a part of its state. It is also termed as a has-a relationship in java.

Composition- Same as Aggregation. There is a strong relationship between the containing object and the dependent object. If you delete the parent object, all the child objects will be deleted automatically.

Advantages of OOPs over Procedure-oriented programming language.

1. OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.
2. OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.
3. OOPs provides the ability to simulate real-world event much more effectively

Object - (State and behaviour)

An object has three characteristics:

- State - It represents the data (value) of an object.
- Behaviour - It represents the behavior (Functionality) of an object such as deposit, withdraw etc.
- Identity - It is used internally by the JVM to identify each object uniquely.

For example - Pen is an object, It's name is Reynolds, color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. - A class is a template or blueprint from which objects are created. So, an object is the instance (result) of a class.

Class - A class is a group of objects which have common properties.

A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and Interface

Instance variable in Java - A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

new keyword in Java - The new keyword is used to allocate memory at run time. All objects get memory in Heap memory area.

Three ways to initialize object.

1. By reference variable
2. By method.
3. By constructor.

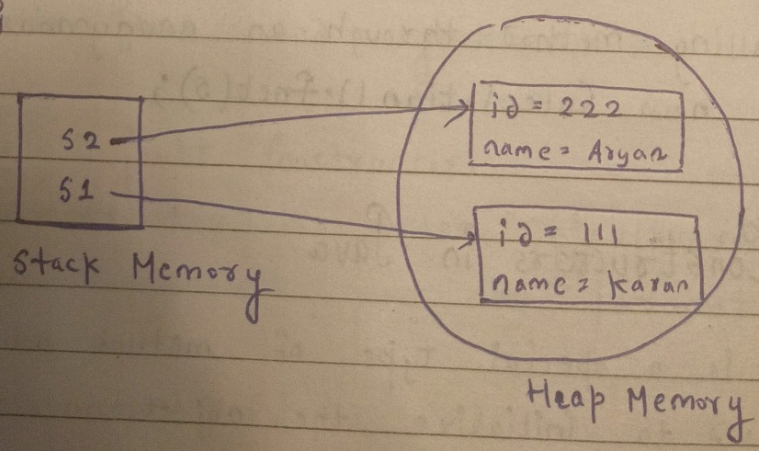
1) By reference Variable -

```
class Student {
 int id;
 String name;
}
class TestS2 {
 public static void main (String[] args) {
  Student s1 = new Student();
      s1.id = 101;
      s1.name = "sheetal";
    System.out.println(s1.id+" "+ s1.name);
} }
```

2) Initialize through method.

Example -

```
class Student {
 int rollno;
 String name;
 void insertRecord(int r, String n) {
    rollno = r;
    name = n;
 }
 void displayInformation() {
   System.out.println(rollno + " " + name); }
class TestStudent4 {
 public static void main (String args[]) {
  Student s1 = new Student();
  Student s2 = new Student();
   s1.insertRecord(111, "karan");
   s2.insertRecord(222, "Aryan");
   s1.displayInformation();
   s2.displayInformation();
 }
}
```



s1, s2 both are reference variable.

What are the different ways to create an object in Java?

- By new keyword
- By newInstance() method
- By clone() method
- By deserialization
- By factory Method

Anonymous Object - It simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

For example -

```
new Calculation();
```

Calling method through a reference -

```
Calculation c = new Calculation();
c.fact(5);
```

Calling method through an anonymous object -

```
new Calculation().fact(5);
```

Constructors in Java.

It is a special type of method which is used to initialize the object.

Every time an object is created using the

new() keyword, atleast one constructor is called.

Note: It is called constructor because it constructs the values at the time of object creation. It is not neccessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any

Two types of constructors

- no-arg constructor
- parameterized constructor

Rules for creating Java Constructor.

1. Constructor name must be the same as its class name.
2. A constructor must have no explicit return type.
3. A java constructor cannot be abstract, static, final, and synchronized.

Java Default Constructor.

A constructor is called "Default constructor" when it doesn't have any parameter.

Syntax -

<class_name>() { }.

Rule :- If there is no constructor in a class, compiler automatically creates a default constructor.

```
class Bike {
}
```

⇒ [Compiler] ⇒

```
class Bike {
    Bike() {}
}
```

What is the purpose of a default constructor?
The default constructor is used to provide the default values to the object like 0, null, etc, depending on the type.

Java Parameterized Constructor.
A constructor which has a specific number of parameters is called parameterized constructor.

Constructor Overloading in Java.
Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.

Example-

```
class Student5 {
    int id;
    String name;
    int age;
```

```
Student5(int i, String n) {
    id = i;
    name = n;
}

Student5(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[]) {
Student5 s1 = new Student5(111, "Karan");
Student5 s2 = new Student5(222, "Aryan", 25);
s1.display();
s2.display();
}
}
```

Difference between constructor and method in Java.

Java Constructor	Java Method.
1. A constructor is used to initialize the state of an object	1. A method is used to expose the behavior of an object.
2. A constructor must not have a return type.	2. A method must have a return type.
3. The constructor is invoked implicitly.	3. The method is invoked explicitly.

4. The java compiler provides a default constructor if you don't have any constructor in a class.	4. The method is not provided by the compiler in any case.
5. The constructor name must be same as the class name.	5. The method name may or may not be same as the class name.

Java Copy Constructor - There is no copy constructor in Java, we can copy the values from one object to another.

There are many ways to copy the values of one object into another in Java are-

1. By Constructor
2. By assigning the values of one object into another.
3. By clone() method of object class.

```
Student6(int i, String n) {
    id = i;
    name = n;
}
Student6(Student6 s) {
    id = s.id;
    name = s.name;
}
```

Que → Does Constructor return any value?
Yes, it is the current class instance, you cannot use return type yet it returns a value.

Que - Can Constructor perform other tasks instead of initialization?
Yes, like object creation, ~~creation~~, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

Que → Is there constructor class in Java?
Yes.

Que → what is the purpose of constructor class?
Java provides a constructor class which can be used to get the internal information of a constructor in the class. It is found in the <u>java.lang.reflect package</u>.

Java static keyword.
The static keyword in Java is used for memory management mainly.

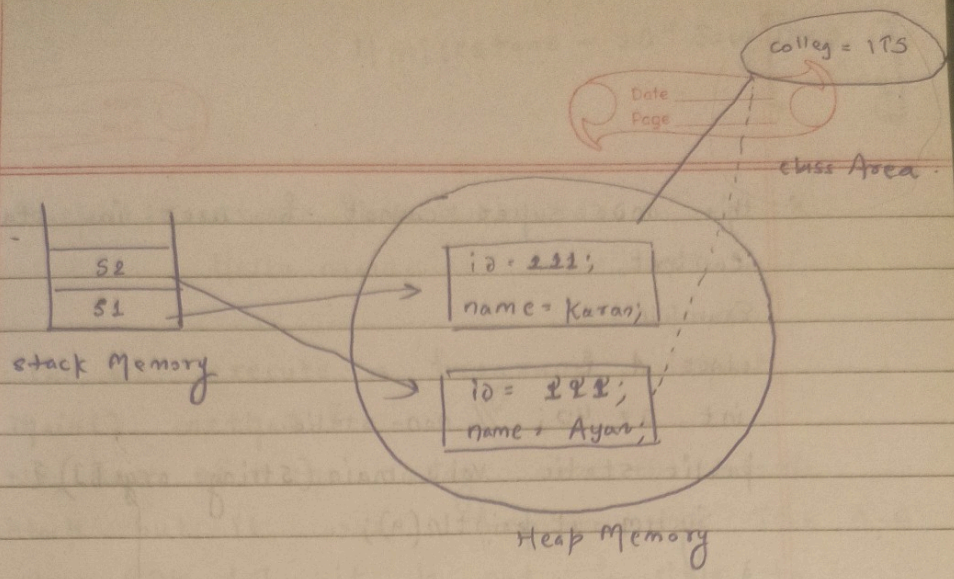
The static can be:
1. Variable
2. Method
3. Block
4. Nested Class

1. Java static variable -

1. The static variable can be used to refer to the common property of all objects (which is not unique for each object).
2. The static variable gets memory only once in the class area at the time of class loading.
3. It makes your program memory efficient (i.e it saves memory).
4. Java static property is shared to all objects.

Example -

```
class Student {
 int rollno;
 String name;
 Static String college = "ITS";

 Student(int r, String n) {
  rollno = r;
  name = n;
 }

 void display() {
  System.out.println(rollno + " " + name + " " + college);}
 }

public class TestStaticVariable1 {
public static void main(String args[]) {
 Student s1 = new Student(111, "Karan");
 Student s2 = new Student(222, "Ayan");
 s1.display();
 } s1.display();
}
```



2. Java Static Method.

1. A static method belongs to the class rather than the object of a class.
2. A static method can be invoked without the need for creating an instance of a class.
3. A static method can access static data member and can change the value of it.

Example -

```
class Calculate {
  static int Cube(int x) {
    return x * x * x;
  }
public static void main (String args[]) {
  int result = Calculate.Cube(5);
  System.out.println (result);
  }
}
```

Output: 125.

Restrictions -

1. The static method can not use non static data member or call non - static method direcly.

2. this and super cannot be used in static context.

Example :-

```
class A {
int a = 40; // non-static.
public static void main (String args[]) {
   System.out.println(a);
}
}
```

Compile Time Error

Why is the Java main method static?
It is because the object is not required to call a static method. If it were a non-static method. JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

3. Java static block.

1. Is used to initialize the static data member
2. It is executed before the main method at the time of classloading.

Example-

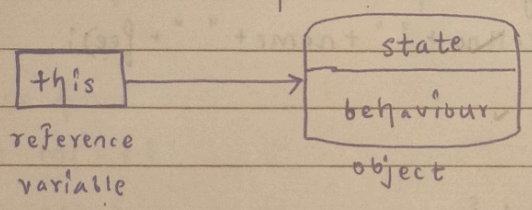
```
class A2 {
static {
System.out.println("static block is invoked"); }
public static void main(String args[]){
System.out.println("Hello main");
}
}
```

Output :- static block is invoked
Hello main

Que- Can we execute a program without main() method ?
No, one of the ways was the static block, but it was possible till JDK 1.6 since JDK 1.7, it is not possible to execute Java class without the main method.

this keyword in Java.

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



Usage of Java this Keyword-

1. "this" can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

1) this: to refer current class instance variable.
If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Example-

```
class Student {
int rollno;
String name;
Float fee;
Student (int rollno, String name, float fee) {
rollno = rollno;
name = name;
fee = fee;
}
void display () {
    SOPln( rollno + " " + name + " " + fee);
}
}
class Testthis1 {
public static void main (String args[]) {
Student s1 = new student (111, "ankit", 5000F);
Student s2 = new student (112, "sumit", 6000F);
s1.display();
s2.display();
}}
```

Output :-

0 null 0.0

0 null 0.0

* In the above example, parameters (formal arguments) and instance variable are same. so, we are using this keyword to distinguish local variable & instance variable.
* If local variables (formal arguments) and instance variables are different, there is no need to use this keyword like in the following program :-

2) this : to invoke current class method.
You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. let's see the example.

```
class A
void m() {}
void n() {
m();  ---> (Compiler) ---> this.m();
}
public static void main(String args[]) {
new A().n();
}}.
```

3) this(): to invoke current class constructor.
The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other

words, it is used for constructor chaining.

Example -

```
class A {
A() { SOPln ("hello a"); }
A(int x) {
this();
SOPln(x);
}
}

class TestThis5 {
psvm (string args[]) {
A a = new A(10);
}}
```

Output :-

hello a

10.

Example-

```
class A {
A() {
this(5);
SOPln ("hello a");
}
A(int x) {
SOPln(x);
}
}

class TestThis6 {
psvm (string args[]) {
A a = new A();
}}
```

Output :-

5

hello a

Real usage of this() constructor call.
This this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

Example -

```
class Student {
int rollno;
String name, course;
Float fee;
Student (int rollno, String name, String course) {
this.rollno = rollno;
this.name = name;
this.course = course;
}
Student (int rollno, String name, String course,
Float fee) {
this(rollno, name, course); // reusing constructor
this.fee = fee;
}
void display() { SOPln(rollno + " "+name+" "+course+" "+fee);}
}
class TestThis7 {
public static void main(String args[]) {
Student s1 = new Student(111, "ankit", "java");
Student s2 = new Student(112, "sumit", "java", 6000f);
```

C.T.E

a = 3
b = 4.

a * (a/b > 1 ? 1 : 0) || b * (b/a > 1 ? 1 : 0)

3 * 0 || 4 * 1 = 4.

4 * 1 > 1.

```
s1.display();
s2.display();
}}
```

* Call to this() must be the first statement in constructor.

4) this: to pass as an argument in the method.

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. let's see the example:-

Example -

```
class S2 {
void m(S2 obj) {
  System.out.Println("method is invoked");
}
void p() {
  m(this);
}
psvm(String args[]) {
 S2 s1 = new S2();
  s1.p();
}
}
```

* Used in Event Handling.

Inheritance in Java (IS-A relationship)

1. Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
2. 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 & fields of the parent class.

Why use inheritance in Java?

- for Method over-riding (so runtime-polymorphism can be achieved)
- for Code Reusability

Terms used in inheritance-

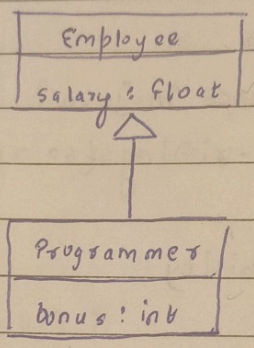
Inheritance

1. Class - A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
2. Sub Class / Child Class - Subclass is a class which inherits th other class. It is also called a derived class, extended class, or child class
3. Super Class / Parent Class :- Superclass is the class from where a subclass inherits the features. It is also called a base class or parent class.
4. Reusability - It is a mechanism which facilitates you to reuse the fields & methods of the existing class when you create a new class.

Syntax of Java Inheritance :-

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

* The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.



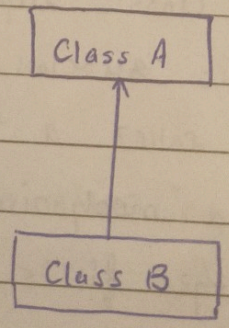
Programmer IS-A Employee.

Types of inheritance in java.

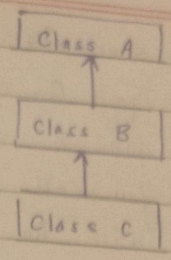
There are three types of inheritance in java : single, multilevel and hierarchical.

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

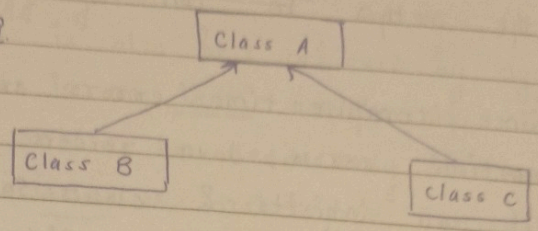
1) Single -



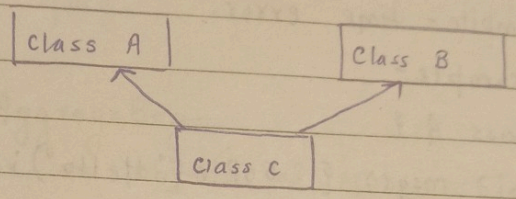
2) Multilevel



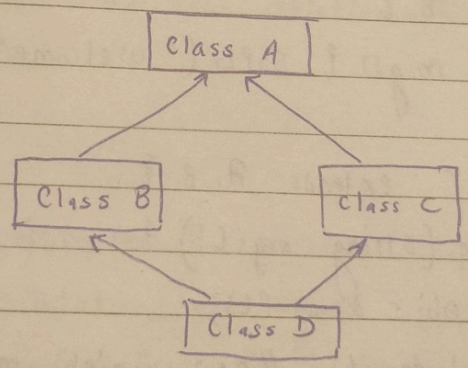
3) Hierarchical



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 ~~to call~~ of A and B class.

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

Example-

```
class A {
void msg() { SOPln("Hello"); }
}
class B {
void msg() { SOPln("welcome"); }
}
class C extends A, B {
psvm (string args[]) {
  C obj = new C();
  obj.msg(); // now which msg() method would be invoked?
}
}.
```

Compile-time error.

Aggregation in Java – If a class have an entity reference, it is known as Aggregation.
Aggregation represents HAS-A relationship.
* Aggregation use for Code Reusability

Why use Aggregation?
- Code reuse is also best achieved by aggregation when there is no is-a relationship
- Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Example of Aggregation.
Employee has an object of Address, address object contains its own information such as city, state country etc. In such case relationship is Employee HAS-A address.

Address.java

```
public class Address {
String city, state, country;
public Address(string city, String state, String country){
    this.city = city;
    this.state = state;
    this.country = country;
}
}
```

Emp.java

```
public class Emp {
int id;
String name;
Address address;

public Emp(int id, String name, Address address){
  this.id = id;
  this.name = name;
  this.address = address;
}

void display() {
 System.out.println(id + " " + name);
 System.out.println(address.city + " " + address.state
      + " " address.country);
}

public static void main(String[] args) {
 Address address1 = new Address("gzb", "UP", "India");
 Address address2 = new Address("gno", "UP", "India");

 Emp e = new Emp(111, "varun", address1);
 Emp e2 = new Emp(112, "arun", address2);

 e.display();
 e2.display();
 }
}
```

Output :-

111 varun
gzb UP india
112 arun
gno UP india

Java Polymorphism .....

if a class has multiple methods having same name but different in parameters, it is known as <u>Method Overloading</u>.

- Method overloading increases the readability of the program.

Different ways to overload the method -

1. By changing number of arguments.
2. By changing the data type.

Note:- In Java, Method overloading is not possible by changing the return type of the method only.

* Method Overloading : Changing no of arguments.

* we are creating "static methods" so that we don't need to create instance for calling methods.

Program-

```
class Adder {
static int add(int a, int b) {
        return a+b;
    }
static int add(int a, int b, int c) {
        return a+b+c;
    }
class TestOverloading1 {
public static void main (String[] args) {
  System.out.println(Adder.add(11, 11));
  System.out.println(Adder.add(11, 11, 11));
}}
```

* Method Overloading : changing data-types of arguments.

```
class Adder {
 static int add(int a, int b) { return a+b;}
 static double add(double a, double b) {
        return a+b; }
}
class TestOverloading2 {
public static void main (String[] args) {
System.out.println(Adder.add(11, 11));
System.out.println(Adder.add(12.3, 12.6));
}}
```

Date
Page

Que- why Method Overloading is not possible by changing the return type of method only?

Ans→ Method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see Example-

```
class Adder {
static int add (int a, int b) { return a+b; }
static double add (int a, int b) { return a+b; }
}
class TestOverloading3 {
public static void main (String [] args) {
System.out.println (Adder.add (11, 11)); Ambiguity
}}
```

Output: Compile Time Error

Que→ Can we overload java main() method?

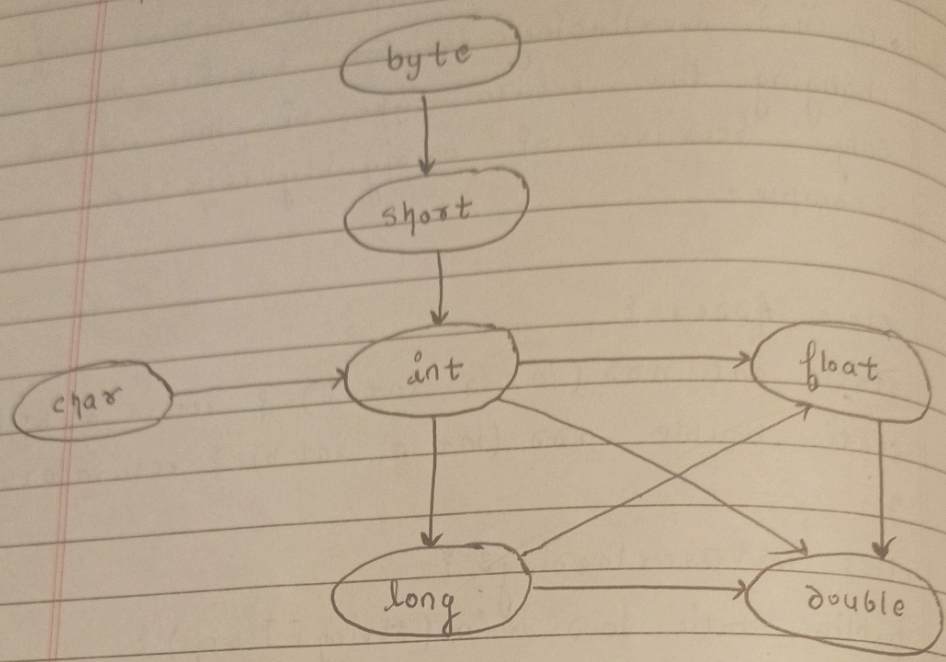
Answer→ Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.

Example -

```
class TestOverloading4 {
public static void main (String [] args) { SOPln (" Hi"); }
public static void main (String args) { SOPln ("Hello"); }
public static void main () { SOPln (" Bye"); }
}
```

Output :- Hi

Method Overloading and Type Promotion -



Example -

```
class OverloadingCalculation1 {
  void sum(int a, long b) { SOPln(a+b); }
  void sum(int a, int b, int c) { SOPln(a+b+c); }
public static void main(String[] args) {
OverloadingCalculation1 obj = new OverloadingCalculation1();
  obj.sum(20, 20); → second int literal will be promoted
  obj.sum(20, 20, 20);        to long
} }
```

Output :-

40

60

long < F loat

Method Overriding in Java

- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.
- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass
- Method overriding is used for runtime polymorphism.

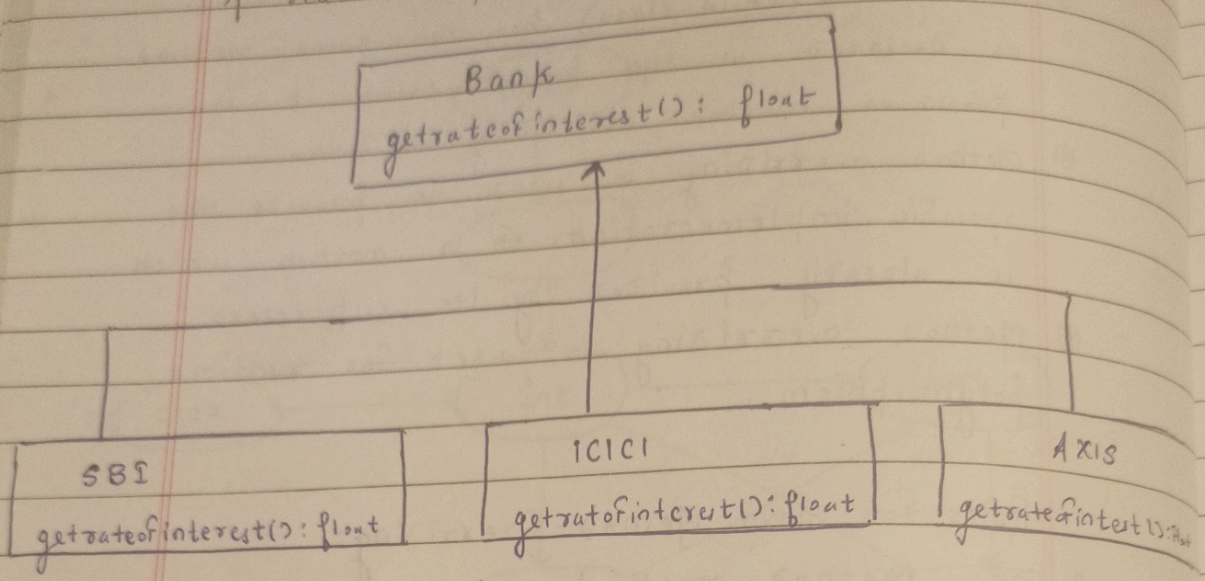
Rules for Java Method-Overriding-

1. The method must have the same name as in the parent class.
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

Example-

```
class Vehicle {
  void run() { System.out.println("Vehicle is running");}
  }
class Bike2 extends Vehicle {
  void run() { System.out.println("Bike is running");}
public static void main (String args[]) {
  Bike2 obj = new Bike2();   // creating object.
  obj.run();  // calling method
}
}
```

A real example of Java Method overriding -



- Java method over-riding is mostly used in Runtime Polymorphism which we will learn in next pages.

Que- Can we override static method?
No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

Que- Why can we not override static method?
It is because the static method is bound with class where as instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

Date
Page

Que - Can we override java main method?
No, because the main is a static method

Method Overloading	Method Overriding
1. Method overloading is used to increase the readability of the program.	1. Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2. Method overloading is performed within class.	2. Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3. In case of method overloading, parameter must be different.	3. In case of method overriding, parameter must be same.
4. Method overloading is the example of compile-time polymorphism.	4. Method overriding is the example of run-time polymorphism.
5. In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading But you must have to change the parameter.	5. Return type must be same or covariant in method-overriding.

Covariant Return Type -
The covariant return type specifies that the return type may vary in the same direction as the sub-class.

* Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type. Let's take a simple example.

Advantages of Covariant Return Type -

1) Covariant return type assists to stay away from the confusing type casts in the class hierarchy and makes the code more usable, readable and maintainable.
2) In this method overriding, the covariant return type provides the liberty to have more to the point return types.
3) Covariant return type helps in preventing the run-time ClassCastException on returns.

Covariant Example-

```
class A1 {
  A1 foo() {
    return this;
    }
  void print() {
    System.out.println("Inside the Class A1");
  }
}
Class A2 extends A1 {
  A2 foo() {
  returun this;
  }
 void print() {
  System.out.println("Inside the class A2");
 }
}
Class A3 extends A2 {
  A3 foo() {
  return this; }
  void print() {
   System.out.println("Inside the class A3");
 } }
public Class CovariantExample {
  public static void main(String[] args) {
  A1 a1 = new A1();
  a1.foo().print();
 A2 a2 = new A2();
```

```
    a2.foo().print();
    A3 a3 = new A3();
    a3.foo().print();
  }
}
```

Output -
Inside the class A1
Inside the class A2
Inside the class A3

Explanations -
In the above program, no typecasting is needed as the return type is specific. Hence, there is no confusion about knowing the type of object getting returned from the method foo(). Also even if we write the code for the 10-15 classes, there would be no confusion regarding the return types of the methods.

Super Keyword in Java

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

Usage of 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.

1) super → Parent class instance variable.

Example -

(It is used if parent class and child class have same fields)

```
class Animal {
String color = "white";
}

class Dog extends Animal {
String color = "black";
void printColor() {
System.out.println(color);
System.out.println(super.color);
}
}

class TestSuper1 {
public static void main(String args[]) {
Dog d = new Dog();
d.printColor(); }}
```

Output :-

black

white

2) super → Invoke parent class method

Example -

```
class Animal {
void eat() {
System.out.println("eating"); }
}
```

```
class Dog extends Animal
void eat() { System.out.println("eating bread...
void bark() { System.out.println("barking ....");}
void work() {
  super.eat();
  bark(); }}
class TestSuper2 {
 public static void main(String args[]) {
 Dog d= new Dog();
   d.work();
 }}
```

(It is used if method is overridden)

Output -

eating.

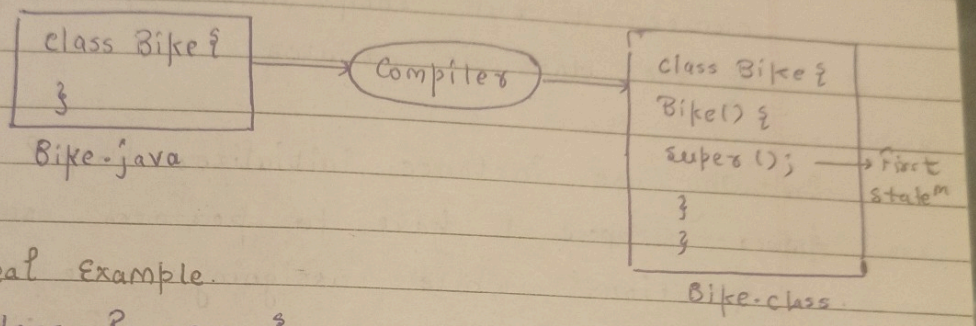
barking...

3) Super → Invoke parent class constructor

```
class Animal {
Animal() { System.out.println("Animal is created");
 }
}

class Dog extends Animal {
Dog() {
super();
System.out.println("dog is created");
}
}

class TestSuper3 {
public static void main(String args[]) {
Dog d = new Dog();
 }}
```

Output -
animal is created
dog is created.

Note:- super() is added in each class constructor automatically by compiler if there is no super() or this().



Real Example.

```
class Person {
 int id;
 String name;
 Person(int id, String name) {
  this.id = id;
  this.name = name;
 }
}

class Emp extends Person {
 float salary;
 Emp(int id, String name, float salary) {
  super(id, name);
  this.salary = salary;
 }

void display() { System.out.println(id+ name+salary);
}.
```

```
class TestSuper5 {
public static void main(String[] args) {
Emp e1 = new Emp(1, "ankit", 45000f);
e1.display(); }}
```

Instance Initializer block.

It is used to initialize the instance data member. It run each time when object of the class is created.

Que - why use instance initializer block?

Answer - suppose I have to perform some operations while assigning value to instance data member e.g a for loop to fill a complex array or error handling etc.

Example -

```
class Bike7 {
 int speed;
 Bike7() { System.out.println("speed is " + speed); }
 { speed = 100; }
public static void main(String[] args) {
 Bike7 b1 = new Bike7();
 Bike7 b2 = new Bike7();
} }
```

Output :-

speed is 100.

speed is 100.

There are three places in java where you can perform operations :-

1. method
2. constructor
3. block.

Que → What is invoked first, instance initializer block or constructor ?

Answer → It seems that instance initializer block is firstly invoked but NO, Instance initializer block is invoked at the time of object creation. The java compiler copies the instance initializer block in the constructor after the first statement super(). so firstly, constructor is invoked, let's understand it by the figure.

```
Class B
B() { SOP("Constructor"); } }
{ SOPln(" Instance Initializer Block"); } }
```

↓

Compiler

↓

```
class B
 B() {
  super();
  { SOPln("instance Initializer Block"); }
   SOPln("constructor"); }
 }
}
```

Rules for instance initializer block -

Three rules -

1) The instance initializer block is created when instance of the class is created.
2) The instance initializer block is invoked after the parent class constructor is invoked
3) The instance initializer block comes in the order in which they appear.

Example of Instance Block -

```
class A {
A() {
System.out.println("parent class constructor");
}
}
class A1 extends A {
A1() {
super();
System.out.println("child class constructor invoked
}
A1(int a) {
super();
System.out.println("child class constructor
                    invoked " + a);
}
{ System.out.println("instance initialized block is
                    invoked "); }
```

```
public static void main(String args[]) {
A1 a1 = new A1();
A1 a2 = new A1(10);
}
}
```

Output :-

parent class constructor invoked
instance initializer block is invoked
child class constructor invoked
parent class constructor invoked
instance initializer block is invoked
child class constructor invoked 10,

Final keyword

The final keyword in java is used to restrict the user. The java final keyword can be used in many context.

1) Variable
2) method
3) class.

1) Java Final Variable

If you make any variable as final, you cannot change the value of final variable (It will be constant)

- Stop Value Change
- Stop Method Overriding
- Stop Inheritance

Example -

```
class Bike9 {
final int speedlimit = 90;
void run() {
speedlimit = 400;
}
public static void main(string[] args) {
Bike9 obj = new Bike9();
obj.run();
}
}.
```

Output -

Compile Time Error.

2) Java final Method.

Example -

```
class Bike {
final void run() { system.out.println("running"); }
}

class Honda extends Bike {
void run() { system.out.println("running safely
        with 100kmph"); }
public static void main(string args[]) {
Honda honda = new Honda();
honda.run();
}
}.
```

Output :-

Compile Time Error.

3) Java Final Class

Example-

```
final class Bike{}
class Honda1 extends Bike{
void run(){ System.out.println("running safely
                with 100kmph");}
public static void main(String[] args){
Honda1 honda = new Honda1();
honda.run();
}}
```

Output-

Compile Time Error.

Que → Is final method inherited?

Yes, final method is inherited but you cannot override it. for example-

```
class Bike {
final void run() { System.out.println("running");}
}
class Honda2 extends Bike {
public static void main(String[] args){
new Honda2().run();
}
}
```

Output-

running

Date
Page

Que→ What is blank or unintialized final variable?
Ans→ A final variable that is not initialized at the time of declaration.

Que- Can we initialize blank final variable?
Ans- Yes, but only in constructor.
Example-

```
class Bike10 {
  final int speedlimit; // blank final variable
  Bike10() {
  speedlimit = 70;
  System.out.println(speedlimit);
  }
  public static void main(string[] args) {
    new Bike1();
  }
```

Output:
70

Que→ static blank final variable
A static final variable that is not initialized ~~only in sl~~ at the time of declaration is known as static blank final variable. It can only be initialized in static block.

Example -

```
class A {
 static final int data;
 static { data = 50; }
 public static void main (String[] args) {
 System.out.println(A.data);
 }
}.
```

Que- What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

Example -

```
class Bike11 {
int cube (final int n) {
 n = n + 2;
 n * n * n;
 }
public static void main (String[] args) {
 Bike11 b = new Bike11();
 b.cube(5);
 }
}
```

Output - Compile Time Error.

Que- Can we declare a constructor final?

No, because constructor is never inherited.

Polymorphism in Java.

It is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 greek words : poly and morphs.

poly → many

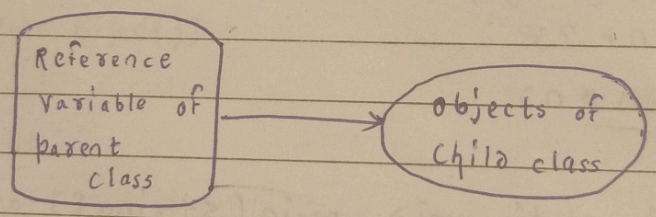
morphs → forms.

Polymorphism

- Runtime polymorphism
  over-riding
- Compile-time polymorphism
  over-loading

Upcasting

If the reference variable of Parent class refers to the object of child class.



Example-

```
class A {}
class B extends A {}
A a = new B(); // upcasting
```

Runtime Polymorphism -

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Java Runtime Polymorphism Example: Bank.

The rate of interest may differ according to banks. For example, SBI, ICICI, and AXIS banks are providing 8.4%, 7.3%, 9.7% rate of interest.

Example-

```
class Bank {
float getRateOfInterest() {
        return 0; }}
class SBI extends Bank {
float getRateOfInterest() {
        return 8.4f; }}
class ICICI extends Bank {
float getRateOfInterest() {
        return 7.3f; }}
class AXIS extends Bank {
float getRateOfInterest() {
        return 9.7f; }}
class TestPolymorphism {
public static void main(String[] args) {
Bank b;
b = new SBI();
System.out.println("SBI " + b.getRateOfInterest());
b = new ICICI();
System.out.println("ICICI " + b.getRateOfInterest());
b = new AXIS();
System.out.println("AXIS" + b.getRateOfInterest());
}
}
```

Date
Page

Java Runtime Polymorphism with Data Member
A member is overridden, not the data members, so runtime polymorphism can't be achieved by data members.

Example -

```
class Bike {
int speedlimit = 90;
}
class Honda3 extends Bike {
 int speedlimit = 150;
}
public static void main(String[] args) {
 Bike obj = new Honda3();
System.out.println(obj.speedlimit);
}
```

Output - 90

Note :- Runtime Polymorphism can't be achieved by data members.

Binding - Connecting a method call to the method body is known as binding.

Static Binding
(Early binding)

Dynamic Binding
(Late Binding)

Static Binding - When type of the object is determined at compiled time.
If there is any private, final, or static method in a class, there is a static binding.

Example -

```
class Dog {
private void eat() { System.out.println("dog is
  eating...."); }
public static void main(String args[]) {
 Dog d1 = new Dog();
    d1.eat();
}
}
```

Dynamic Binding - when type of the object is determined at run-time.

Example -

```
class Animal {
void eat() { SOPln("animal"); }
}
class Dog extends Animal {
void eat() { SOPln("dog is created"); }
public static void main(String[] args) {
Animal a = new Dog();
 a.eat();
}
}
```

Output - dog is eating.

* Object have a type-

```
class Animal { }
class Dog extends Animal {
public static void main (String[] args) {
  Dog d1 = new Dog(); }
}
```

* An object is an instance of particular java class, but it is also an instance of its superclass.

→ d1 is an instance of Dog class, but it is also an instance of Animal.

Java instanceof operator.

The java instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

Example-

```
class Simple1 {
public static void main(String args[]) {
Simple1 s = new Simple1();
```

```
System.out.println(s instanceof Simple1); // true
}
}
```

Example -

```
class Dog2 {
public static void main(String[] args) {
  Dog2 d = null;
System.out.println(d instanceof Dog2); // false
}
}
```

Downcasting with java instanceof operator

When Subclass type refers to the object of the parent class it is known as downcasting.

```
Dog d = new Animal(); // Compilation error.
```

* If we perform downcasting by typecasting, <u>ClassCastException</u> is thrown at run-time.

```
Dog d = (Dog) new Animal();
```

Abstraction. - It is a process of hiding the implementation details and showing only functionality to the user.
for example → sending sms where you type the text and send the message. You don't know the internal processing about the message delivery

Two ways to achieve abstraction

1) Abstract class

2) Interface

Abstract class in Java.

- 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 constructor & static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

For example -

abstract class A{}.

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as abstract method.

example -

abstract void printStatus();

Example -

Note :- Mostly we don't know about the implementation class (which is hidden to the end user) and an object of the implementation class is provided by the factory method

Factory Method - It is a method that returns the instance of the class.

Example :-

```
abstract class Shape {
abstract void draw();
}

class Rectangle extends Shape {
void draw() { System.out.println("drawing rect"); }
}

class Circle1 extends Shape {
void draw() { System.out.println("drawing circle"); }}
class TestAbstract {
public static void main(String[] args) {
  Shape s = new Circle1();
    s.draw(); } }
```

Output :- drawing circle.

Rule :- If there is an abstract method in a class, that class must be abstract.

Interface in Java

1. An interface in java is a blueprint of a class. It has static constants and abstract methods.

2. Interface also represents the IS-A relationships.

3. It is used to achieve abstraction and multiple inheritance in java.

4. Since, Java 8, we can have default and static methods in an interface.

5. Since Java 9, we can have private methods in an interface.

Why use Java Interface?

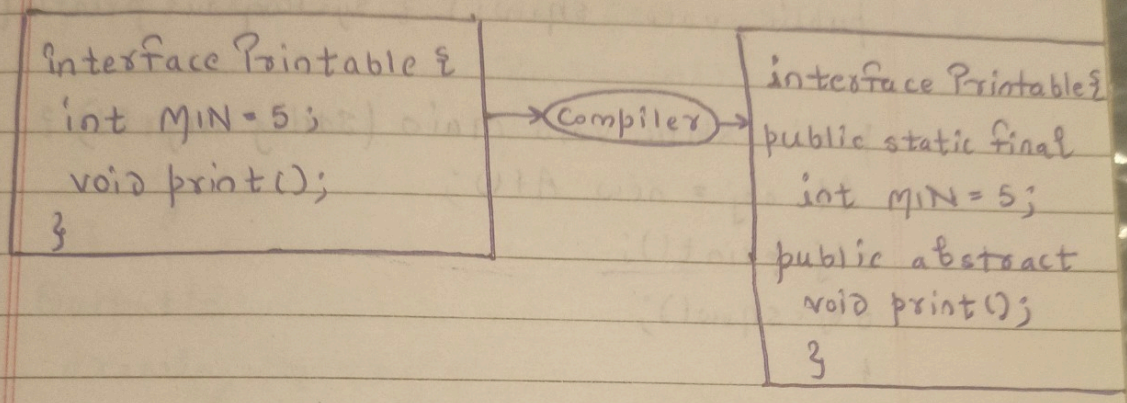
There are mainly three reasons to use interface -

i) It is used to achieve abstraction.

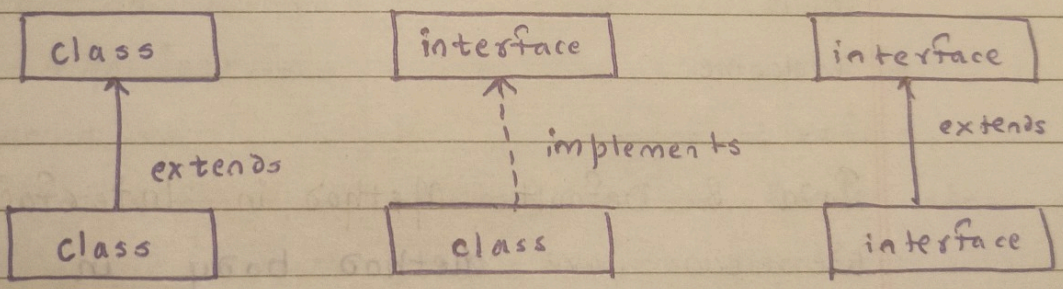
ii) By interface, we can support the functionality of multiple inheritance.

iii) It can be used to achieve loose coupling.

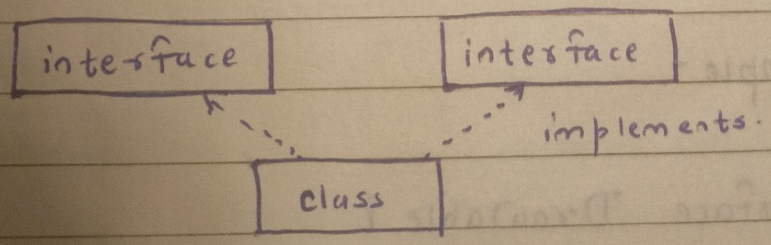
Internal addition by the compiler-
The java compiler adds public and abstract keywords before the interface methods.
Moreover, it adds public, static and final keywords before data members.



Relationship between classes and interfaces.



Multiple inheritance in Java by interface.



Code:-

```
interface Printable {
  void print();
}
```

```
interface Showable {
  void show();
}
class A7 implements Printable, Showable {
public void print() { System.out.println("Hello");
public void show() { System.out.println("welcome");

public static void main (String args[]) {
 A7 obj = new A7();
obj.print();
obj.show();
}
}
```

Output →
Hello
Welcome.

Java 8 Default Method in Interface.
We can have method body in interface. But we need to make it default method.

Example -

```
interface Drawable {
void draw();
default void msg() {
System.out.println("default method");
} }
```

```
class Rectangle implements Drawable {
public void draw() {
    System.out.println("drawing rectangle"); }
}
class TestInterfaceDefault {
public static void main (String[] args) {
Drawable d = new Rectangle();
d.draw();
d.msg();
}}
```

Output:-

drawing rectangle

default method.

Que- What is marker or tagged interface?

Answer → An interface which has no member is known as a marker or tagged interface. for example, Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

Abstract	Interface.
1. Abstract class have abstract and non-abstract methods.	1. Interface can have only abstract methods. Since Java 8, it can have default & static method.
2. Abstract class doesn't support multiple inheritance.	2. Interface supports multiple inheritance.
3. Abstract class can have final, non-final, static and non-static variables.	3. Interface has only static & final variables.
4. Abstract class can provide the implementation of interface.	4. Interface can't provide the implementation of abstract class.
5. The abstract keyword is used to declare abstract class.	5. The interface keyword is used to declare interface.
6. An abstract class can extend another Java class and implement multiple Java interfaces.	6. An interface can extend another java interface only.
7. An abstract class can be extended using keyword "extends".	7. An interface can be implemented using keyword "implements".
8. A java abstract class can have class members like private, protected etc.	8. Members of a java interface are public by default.
9. public abstract class Shape { public abstract void draw(); }	9. public interface Drawable { void draw(); }

Note:-

face.
an have only
hods since
can have
tic method
pports
itance.
only static
des
t provide
ation of

keyword
lare interface
a extend
terface only
n be
g keyword

va
lic by

Drawable{

Note:- Abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%)

Program-

```
interface A{
void a();
void b();
void c();
void d();
}

abstract class B implements A{
public void c(){ system.out.println("I am c"); }
}

class M extends B{
public void a(){ SOPln("I am a"); }
public void b(){ SOPln("I am b"); }
public void d(){ SOPln("I am d"); }
}

class Test5{
public static void main(String[] args){
A a = new M();
a.a();
a.b();
a.c();
a.d();
}}
```

Output:-

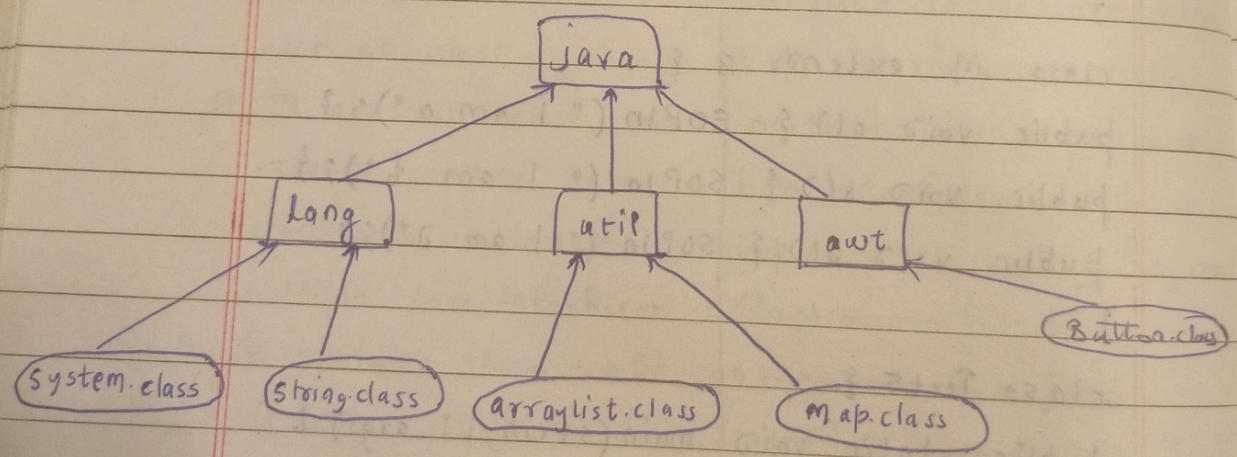
1. I am a
2. I am b
3. I am c
4. I am d

Java Package-

1. A java package is a group of similar types of classes, interfaces and sub-packages.
2. Package in java can be categorized in two form, built-in package and user-defined package.

Advantage of Java Package-

1. Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2. Java package provides access protection.
3. Java package removes naming collision.



How to compile java package-

javac -d directory javafilename

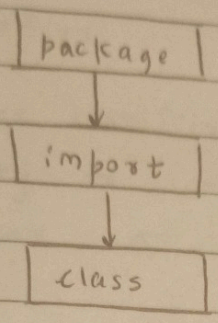
→ javac -d Simple.java

How to run package-

java mypack.Simple

How to access package from another package?

1. import package.*;
2. import package.classname;
3. Fully qualified name.



Note:- The standard of defining package is domain.company.package.

or/and

Date
Page

Access Modifiers in Java.

There are two types of modifier.

access modifier

non-access modifier

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.

Four types of Java access modifiers -

1. Private
2. Default
3. Protected
4. Public

	Within class	within package	outside package	by subclass outside
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	N	Y
Public	Y	Y	Y	Y

1. Private

A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

Role of Private Constructor -

If you make any class constructor private, you

cannot create the instance of that class from outside the class.

Note:- A class cannot be private or protected except nested class.

2) Default

- If you don't use any modifier, it is treated as default by default.
- The default modifier is accessible only within package. It cannot be accessed from outside the package.
- It provides more accessibility than private.

3) Protected

- The protected access modifier is accessible within package and outside the package but through inheritance only
- It cannot be applied on the class.

Example-

```
package pack;
public class A {
  protected void msg() { system.out.println("Hello");
}
package mypack;
import pack.*;
class B extends A {
public static void main(string[] args) {
  B obj = new B();
  obj.msg();
}
```

Date
Page

Java Access Modifiers with Method Overriding-

If you are overriding any method, overridden method (i.e declared in subclass) must not be more restrictive.

```
class A {
protected void msg() { System.out.println("Hello"); }}
public class Simple extends A {
  void msg() { System.out.println("Hello"); } // C.T error
public static void main(String[] args) {
  Simple obj = new Simple();
    obj.msg();
}
}.
```

Encapsulation in Java.

It is a process of wrapping code and data together into a single-unit. for example a capsule which is mixed of several medicines.

- We can create a fully encapsulated class in Java by making all the data members of the class private

Ex- JavaBean is fully encapsulated class.

Advantage of Encapsulation.

1) By providing only a setter or getter method, you can make the class read-only or write-only

2.) It provides you the control over the data

3.) It is a way to achieve data hiding in Java.

4.) The encapsulate class is easy to test.

Example -

```
package com.javatpoint;
public class Student {
  private String name;
  public String getName() {
     return name;
  }
public void setName (String name) {
        this.name = name;
}
}
```

```
package com.javatpoint;
class Test {
public static void main (String [] args) {
  Student s = new Student();
    s.setName ("vijay");
  System.out.println (s.getName());
```

Example -

Read-only class.

```
public class Student {
private String College = "AKG";
public String getCollege() {
      return college;
}
}.
s.setCollege("KITE") // compile Time error.
```

Write-Only class

Object Class in Java

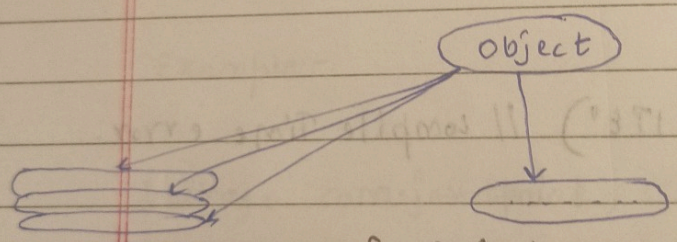
The Object class is the parent class of all the classes in Java by default.

- The Object class is beneficial if you want to refer any object whose type you don't know.

Example-

Object obj = getObject();

there is getObject() method that returns an object but it can be of any type like Employee, Student etc.



Methods of Object class -

1. public final Class getClass() → returns the Class class object of this object. The Class class can further be used to get the metadata of this class.

2. public int hashcode() → returns the hashcode number for this project.

Object Cloning in Java.

The object cloning is a way to create exact copy of an object. The clone() method of Object class is used to clone an Object.

- The java.lang.Cloneable interface must be implemented by the class whose Object clone we want to create.

- If we don't implemented Cloneable interface, clone() method generates ClassNotSupportedException.

- The Clone() method is defined in the Object class. Syntax of the clone() method is as follows.

Syntax -

protected Object clone() throws CloneNotSupportedException.

Why use clone() method?

The clone() method saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time ~~of~~ to

be performed that is why we use object cloning.

Advantages	Disadvantages
1. Don't need to write lengthy and repetitive codes.	1. Lot of syntaxes to our codes.
2. It is the easiest and most efficient way of copying objects,	2. Implement Cloneable interface.
3. Clone() is the fastest way to copy array.	3. Object.clone() is protected.
	4. It doesn't invoke any constructor so we don't have any control over object construction.

Example of clone() method.

```
class Student18 implements Cloneable {
 int rollno;
 String name;

Student18(int rollno, String name) {
 this.rollno = rollno;
 this.name = name;
}
```

```
public Object clone() throws CloneNotSupportedException {
    return super.clone();
}

public static void main(String[] args) {
 try {
 Student18 s1 = new Student18 (101, "amit");
 Student18 s2 = (Student18) s1.clone();

 System.out.println (s1.rollno + " " + s1.name);
 System.out.println(s2.rollno + " " + s2.name);
 }
 catch (CloneNotSupportedException c) { }
 }
}
```

Output :-

```
101 amit
101 amit
```

Date
Page

Java Math class
It provides several methods to work on math calculations like min(), avg(), max(), sin(), cos(), tan(), round(), ceil(), floor(), abs() etc.

Wrapper classes in Java
The wrapper class in Java provides the mechanism to convert primivite into object & object into primitive.

1. The automatic conversion of primitive into an object is known as autoboxing
2. Vice-versa mean object into primitive is unboxing.

Use of wrapper classes in Java

1. Change the value in Method - Java support only call by value. so if we pass a primitive value, it will not change the original value. But if we convert the primitive value in an object, it will change the original value.

2. Serialization - We need to convert the objects into streams to perform the serialization. if we have a primitive value, we can convert it in objects.

through the wrapper classes.

3. Synchronization - Java Synchronization works with objects in multithreading.

4. java.util package - The java.util package provides the utility classes to deal with objects.

5. Collection Framework - Java collection framework works with object only.

The eight classes of the java.lang package are known as wrapper classes in Java.

Primitive Type	Wrapper classes
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
Double	Double

Note:- Java 5, we do need the use of valueOf() method of wrapper classes to convert the primitive into objects

Autoboxing Example -
Primitive ⟶ Corresponding wrapper class

Example :-

```
public class Wrapper {
  public static void main (String [] args){
    int a = 20;
    Integer i = Integer.valueOf(a);
    Integer j = a;
    SOP(a + " " +i+ " " +j);
}}
```

Output :-
20 20 20.

Unboxing Example.
Wrapper type ⟶ Primitive type

Example :-

```
public class Wrapper {
public static void main (String args[]){
  Integer a = new Integer(3);
  int i = a.intValue();
  int j = a;
  SOP(a+" " +i+ " " +j);
}}
```

Output :- 3 3 3

Custom Wrapper class in Java -
Java Wrapper classes wrap the primitive data types, that is why it is known as wrapper classes. We can also create a class which wraps a primitive data type.

Example -

```
class Javatpoint {
 private int i;
 Javatpoint() { }
 Javatpoint(int i) {
   this.i = i;
 }
public int getvalue() {
   return i;
 }

public void setvalue(int i) {
   this.i = i;
 }
@override
public String toString() {
   return Integer.toString(i);
 }
}

public class TestJavapoint {
public static void main (String[] args) {
Javatpoint j = new Javatpoint(10);
```

```
    SOP(i);
}}
```

Output :-
10

Call by Value and Call by Reference in
There is one call by value in Java,
not call by reference. If we call
a method passing a value, it is
known as call by value. The
changes being done in the method, is
not affected in the calling method.

Example - 1

```
class Operation {
  int data = 50;

  void change (int data) {
    data = Data + 100;
  }

public static void main (String [] args) {
  Operation op = new Operation ();

  SOP(" befor change " + op.data);
  op.change (500);
  SOP(" after change " + op.data);
}}
```

Output :-

before change 50

after change 50

Example - 2

```
class Operation {
  int data = 50;

  void change (operation op) {
    op.data = op.data + 100;
  }

  public static void main (String[] args) {
    Operation op = new operation();

    SOP ("before change " + op.data);
    op.change (op);
    SOP (" after change " + op.data);
  }
}
```

Output :-

before change 50

after change 150

Java strictfp keyword -

- Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable.

Legal code for strictfp keyword -
The strictfp keyword can be applied on methods, classes and interfaces.

1. strictfp class A { }
2. strictfp interface M { }
3. class A {
   strictfp void m() { }
   }

Illegal code for strictfp keyword
The strictfp keyword cannot be applied on abstract methods, variables & constructors.

1. class B {
   strictfp abstract void m();
   }
2. class B {
   strictfp int dat = 10;
   }
3. class B {

```
strictfp B() {}
}.
```

Creating API Document

- we can create document api in java by the help of javadoc tool.
- In the java file, we must use the documentation comment /**....*/ to post infor^n for the class, method, constructor etc.

Example :-

```
package com.abc;
public class M {
  public static void cube(int n) {
      System.out.println(n*n*n); }
}
```

- To create the document API, you need to use the javadoc tool followed by java file name. There is no need to compile the java file.

```
javadoc M.java.
```

Date
Page

Java Command line Arguments.

The java - command-line argument i.e passed at the time of running the java program.

Example -

```
class CommandLineExample {
public static void main (string [] args) {
  SOPln (" your first argument is :" arg[0]);
}
}
```

Compile by > javac CommandLineExample.java
run by > java CommandLineExample sonoo.

Output -
Your first argument is : sonoo.

Object	Class
1. Object is an instance of a class.	1. Class is a blueprint or template from which objects are created.
2. Object is a real-world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	2. Class is a group of similar objects.
3. Object is a physical entity.	3. Class is a logical entity.
4. Object is created through new keyword mainly e.g. Student s1 = new Student();	4. Class is declared using class keyword eg. class Student {}
5. Object is created many times as per requirement.	5. Class is declared once.
6. Object allocates memory when it is created.	6. Class doesn't allocate memory when it is created.
7. There are many ways to create object in java such as new keyword, newInstance() method, clone() method, factory method() and deserialization.	7. There is only one way to define class in java using class keyword.

Real life example-

1. Class : Human, Object : Man, Woman.
2. Class : Fruit, Object : Apple, Banana, Mango, Guava etc.
3. Class : Mobile Phone, Object : iPhone, Samsung, Moto.
4. Class : Food, Object : Pizza, Burger,

Exception Handling in Java.

What is Exception in Java?

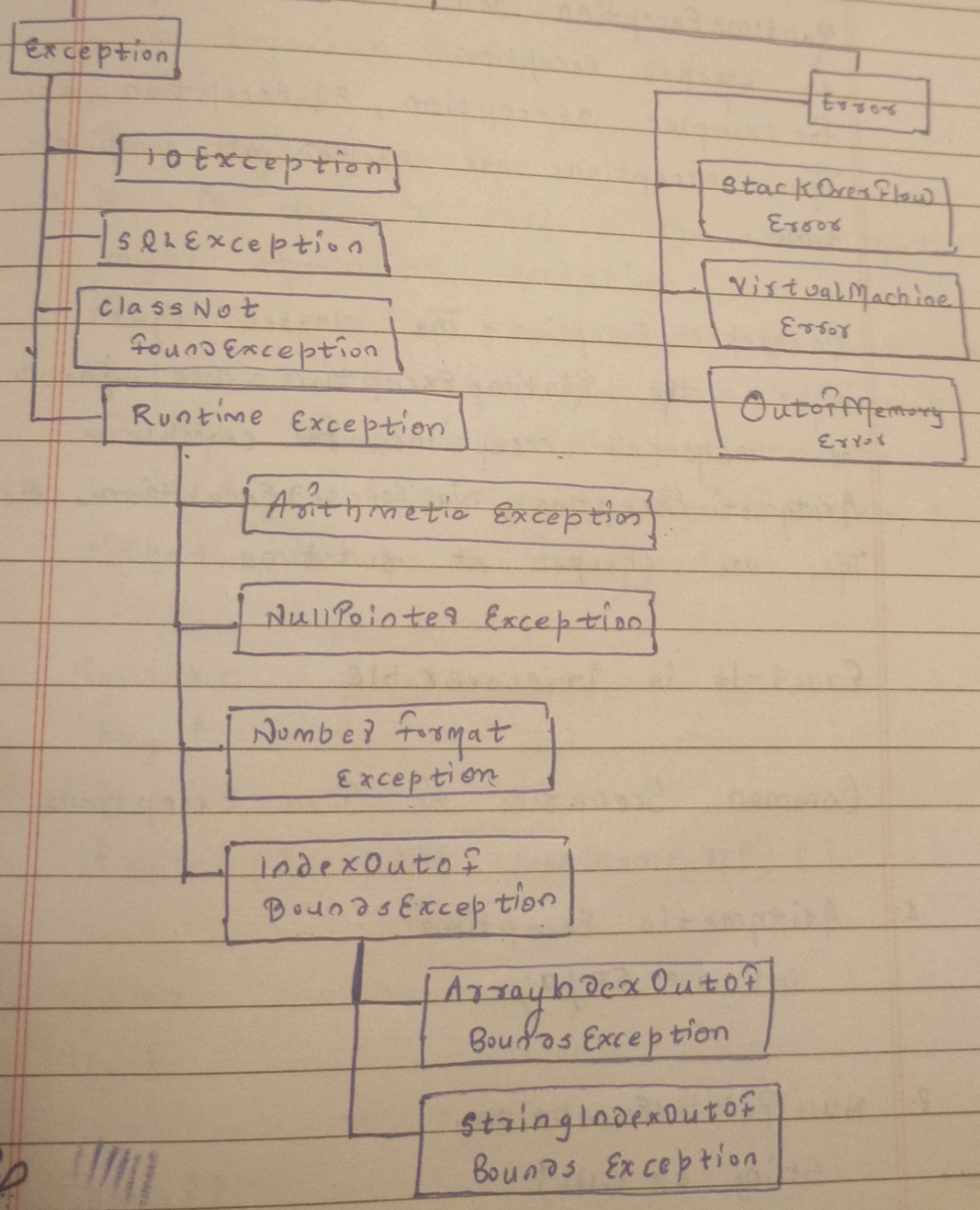
- Exception is an abnormal condition.
- In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at run-time.

What is Exception Handling in Java?
It is a mechanism to handle runtime errors, such as ClassNotFoundException, IOException, SQLException, RemoteException etc.

In interview
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.

Hierarchy of Java Exception Classes -

- The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses.



Types of Exception Handling

- Checked Exception
- Unchecked Exception
- Error

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

Unchecked Exception - The classes that inherit the RuntimeException are known as unchecked exceptions. for example - ArithmeticException, NullPointerException, etc.
They are checked at run-time.

Error - It is irrecoverable.

Common Scenarios of Java Exceptions.

1. Arithmetic Exception.
```
int a = 50/0;
```

2. Null PointerException.
```
String s = null;
SOPln(s.length());
```

3. NumberFormatException.

```
String s = "abc";
int i = Integer.parseInt(s);
```

4. ArrayIndexOutOfBounds.

```
int a[] = new int[5];
a[10] = 50;
```

Java try-catch block.

1. Java try block.

- It is used to enclose the code that might throw an exception.
- If an exception occurs at the particular statement in the try block, the rest of the block-code will not execute.
- Java try block must be followed by either catch or finally block.

Syntax -

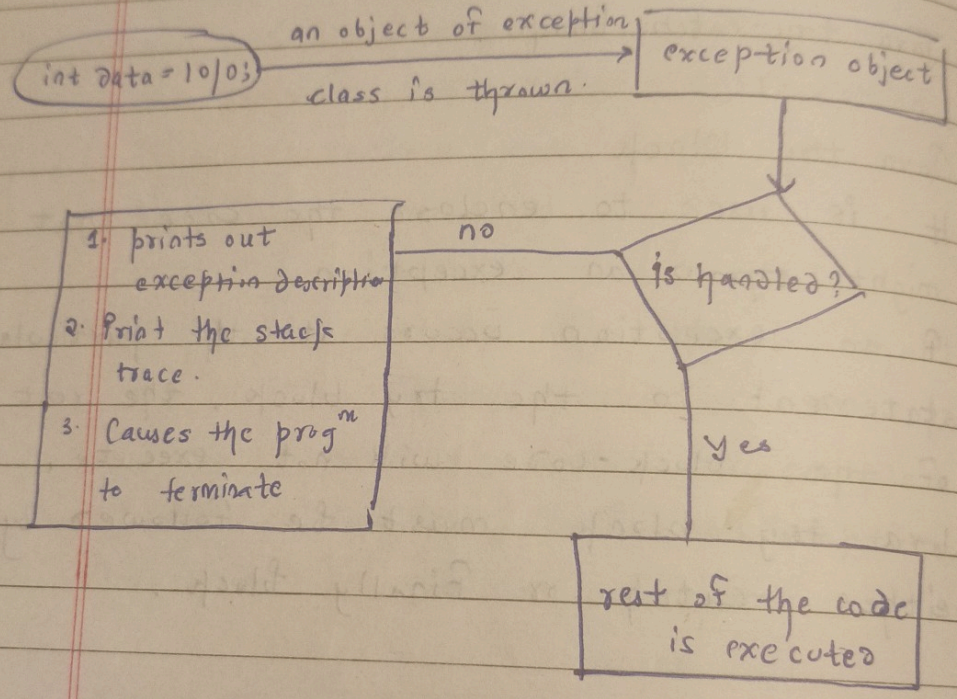
```
try {

} catch (Exception_class_name ref) { }.
```

Syntax -

```
try {
}
finally { }
```

Java catch block -
It is used to handle the Exception by declaring the type of exception within the parameter.

Internal Working of Java try-catch block -



Example -

```
Ques 1- try {
        int data1 = 50/0; ←
    }
    catch (Exception e) {
        int data2 = 50/0; ←
    }
    SOP ("rest of the code");
```

Output -
Exception in thread "main" java.lang.ArithmeticException: / by zero

Ques- try {
int data = 50/0;
}
catch (ArrayIndexOutOfBoundsException e) {
SOPln(e);
}
SOP("rest");
}
}

Output -
Exception in thread "main" java.lang.ArithmeticException: / by zero.

Java Multiple Catch Block-

Note :-
1. At a time only one exception occurs and at a time only one catch block is executed.
2. All catch blocks must be ordered from most specific to most general i.e catch for ArithmeticException must come before catch for Exception.

Que 1- try {
int a[] = new int[5];
a[5] = 30/0;
}
catch (ArithmeticException e) {
SOPln("Arithmetic Exception occurs");
}
catch (ArrayIndexOutOfBoundsException e)
SOPln("ArrayIndex OutofBound occurs");
}
catch (Exception e) {
SOPln("Parent");
}
SOPln("rest of the code");
}
}

Output :-
Arithmetic Exception occurs.
rest of the code.

Que 2- try {
String s = null;
SOPln(s.length());
}
catch (ArithmeticException e) {
SOPln("Arithmetic occurs");
}
catch (ArrayIndexOutOfBoundsException e) {

```
SOPln(" ArrayIndexOutOfBounds occurs");
}
catch(Exception e) {
SOPln("Parent Exception occurs");
}
SOP("rest of the code");
}
}
```

Output -
Parent Exception occurs
rest of the code

Que 3 -
```
try {
int a[] = new int[5];
a[5] = 30/0;
}
catch(Exception e) { SOPln(" common
task completed"); }
catch(ArithmeticException e) {
SOPln(" task1 is completed"); }
catch(ArrayIndexOutOfBoundsException e) {
SOPln(" task2 is completed"); }
SOPln("rest of the code");
}
}
```

Output :-
Compile-time-error.

Java Nested try block

Example -

```
public static void main(String[] args){
 try {
  try {
   try {
      int[] arr = {1, 2, 3, 4};
      SOPln (arr[10]);
   }
   catch(ArithmeticException e) {
      SOPln("inner block 1");
   }
  }
  Catch(ArithmeticException e) {
      SOPln("inner block 2");
  }
 }
 catch(ArrayIndexOutofBoundsException e4){
      SOPln("outer");
 }
 catch (Exception e5) {
   SOP("Exception");
   SOP(" handled in main");
 }
}
}
```

Java Finally Block - It is a block used to execute important code such as closing the connection, etc.

- Java finally block is always executed whether an exception is handled or not.
- The finally block follows the try-catch blocks.

Why use Java Finally block?

- Finally block in Java can be used to put 'cleanup' code such as closing a file, closing connection etc.
- The important statements to be printed can be placed in the finally block.

Usage of Java Finally

- when an exception does not occur
- when an exception occur but not handled by the catch block
- when an exception occurs and is handled by the catch block

Example -

```
public class Test {
    public static void main(String args[]) {
    try {
        System.out.println("Inside try block");
        int data = 25/0;
```

```
System.out.println(data);
}
catch (ArithmeticException e) {
System.out.println("Exception handled");
System.out.println(e);
}
finally {
System.out.println("Finally block executed");
}
System.out.println("rest of the Code");
}
}
```

Output :-

Inside try block
Exception handled
java.lang.ArithmeticException : / by zero
Finally block is alway executed
rest of the code

Rule :- 1. For each try block there can be zero or more catch blocks, but only one finally block.

2. The finally block will not be executed if the program exit (either by calling System.exit(), or by causing a fatal error that causes the process to abort.

Java Throw Exception

- The Java throw keyword is used to throw an exception explicitly.

The syntax of the java throw keyword-

```
throw new IOException("sorry device error");
```

Example-1: Throwing Unchecked Exception

```
public class Test {
 public static void validate(int age) {
     if(age<18) {
        throw new ArithmeticException("Person");
     }
     else {
        System.out.println("Person is eligible");
     }
 }
 public static void main(String[] args) {
     validate(13);
     System.out.println("rest of the code");
 }
}
```

Output :-

Exception in thread "main" java.lang.ArithmeticException : Person is not eligible to vote.

Note:- If we throw unchecked exception from a method, it is must to handle the exception or declare in throws clause.

Note:- If we throw a checked exception using throw keyword, it is must to handle the exception using catch block or the method must declare it using throws declaration.

Example-2: Throwing Checked Exception

```
public class Test {
  public static void method() throws
            FileNotFoundException {

  FileReader File = new FileReader("C:\\Users
  BufferedReader FileInput = new BufferedRead

  throw new FileNotFoundException();
}

public static void main(String[] args) {
  try {
      method();
  }
  Catch (FileNotFoundException)
    { e.printStackTrace();
    }

  System.out.println("rest of code");
  }
}
```

xception from a
andle the
ws clause

xception using
to handle
block or the
ng throws

ception

throws

("C:\\User");
BufferedReader(fr);

;

args) {

Output -

```
java.io.FileNotFoundException
rest of the code
```

Example - Throwing User-defined Exception.

```
class UserDefinedException extends Exception
{
    public UserDefinedException(string str) {
        super(str);
    }
}
public class Test {
    public static void main(String args[]) {
        try {
            throw new UserDefinedException("This User");
        }
        catch(UserDefinedException ude) {
            SoPln("Caught the exception");
            SoPln(ude.getMessage());
        }
    }
}
```

Output :-

Caught the exception
This is user

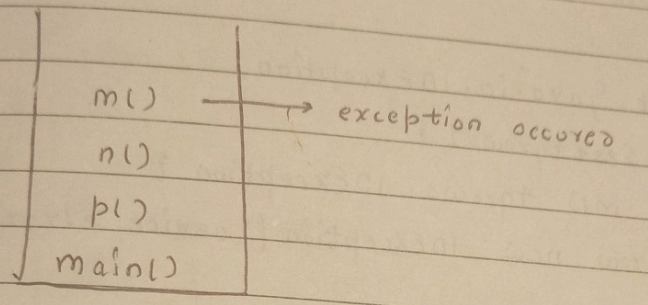
Exception Propogation - An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method. If not caught there, the exception agains drops down to the previous method, so on until they are caught or until they reach the very bottom of the call stack.

Example-

```
class Test {
  void m() {
    int data = 50/0;
  }
  void n() {
    m();
  }
  void p() {
  try {
    n();
  } catch (Exception e) {
  System.out.println("exception handled"); }
}

public static void main (String[] args) {
  Test obj = new Test();
  obj.p();
  SOPln("normal flow");
} }
```

Output :-

exception handled
normal flow



Note :- By default, Unchecked Exceptions are forwarded in calling chain
By default, checked Exceptions are not forwarded in calling chain

Java throws keyword.
It is used to declare an exception.

Syntax of Java throws
return_type method_name() throws exception class {

}

What exception should be declared?
Checked exception only, because.

- Unchecked exception - under our control so we can correct our code.

error - beyond our control. for example - we are unable to do anything if there occurs Virtual Machine Error and etc.

Example-

```
import java.io.IOException;
class Testthrows1 {
  void m() throws IOException {
    throw new IOException("device error");
  }
  void n() throws IOException {
    m();
  }
  void p() {
  try {
    n();
  } catch(Exception e) {
    System.out.println("exception handled"); }
}
public static void main(String[] args) {
  Testthrows1 obj = new Testthrows1();
    obj.p();
  System.out.println("normal flow...");
}
}
```

Output:-

exception handled

normal flow...

example -
thing if
nor and etc.

error");

);

There are two Cases -

Case 1: Handle Exception Using try-catch block

```
class M {
void method() throws IOException {
  throw new IOexception("device error");
}
}
public class Testthrow {
  public static void main(String[] args) {
  try {
  M m = new M();
  m.method();
  } catch (Exception e) {
    System.out.println("exception handled"); }

  System.out.println("normal flow...");
}
}
```

Output :-
exception handled
normal flow...

Case 2: Declare Exception.

(A) If exception does not occur.

Example -

```
class M {
void method() throws IOException {
System.out.println("device operation");
}
}
Class TestThrows {
public static void main (String[] args) throws
IOException {
M m = new M();
m.method();
SoPln ("normal flow");
}
}
```

Output :-

device operation

normal flow

(B) If exception occurs.

Example -

```
class M {
void method() throws IOException {
throw new IOException ("device");
} }
```

```
class TestThrows {
  psvm(String[] args) throws IOException {
    M m = new M();
    m.method();
    SOPln("normal Flow");
  }
}
```

Output -

device error.