Core Java

What is Java?

Java is a high-level, object-oriented programming language.
It is designed for portability, allowing applications to run on any device with a Java Virtual Machine (JVM).

Features of Java: **SOPPSRM**

Simple :

-Simple to learn and use(Syntax's).

Object-Oriented :

Java is an object-oriented programming language. Everything in java is an object. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are : APIE

1. Object
2. Class
3. Abstraction
4. Polymorphism
5. Inheritance
6. Encapsulation

Portable :

-Can be used with any other language

Platform independent :

- ➢ Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc.
- ➢ Java code is compiled by the compiler and converted into byte code.
- ➢ This byte code is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

Secured :

- ➢ Any problem happens only the JVM(JAVA VIRTUAL MACHINE) will get affected but the operating system is safe.

Robust :

- ➢ Java is considered robust due to its strong type checking, automatic memory management, and built-in exception handling,

which reduce errors and enhance stability.

Multithreaded :

- ➢ Java is multithreaded because it allows multiple threads to run concurrently within a program,which improves performance, responsiveness, and scalability.

BASIC STRUCTURE OF PROGRAM

```
public class Demo……(1){
    public static void main(String args[ ] )..........(2)
    {
        System.out.println("This is my first program");.....(3)
    }
}
```

Every programme has 3 main parts

1.CLASS DECLARATION

Ex: public class Demo

It consists of 3 things

1. Access Modifier :

- It indicates that program is accessible to other users or not
- There are 4 access modifiers in java public,private,protected and default
- in above section class is public so it is freely accessible
- All access modifiers will be in lower case.

2. class :

- class is a keyword(reserved word or predefined words) in java.
- Every program must start with a class keyword - all keywords must start with a smaller case so class-c is small.

3. class Name :

- Every class has some name i.e class name or program name or file name
- class name for standard should start with Capital letter
- Java File name and class name must be same for remembering purpose
- class name can only be combination of A-Z,a-z,0-9,$,_ Note:{//}--------->scope of class

2.DEFINING MAIN METHOD

Ex: public static void main(String args[ ])

{
/ / LOGIC OF APP
}

It consists of 5 parts

1.Access Modifier :

2.Non Access Modifier :

static----------> can be used without object creation

non static---------> needs to be used with object creation

3.return type :

void indicated no specific return type

4.method name :

-if any word contains( )----> we can identified it as a method Ex: main( ),run( ),display( ) etc

-main is name of method

3.PRINTING STATEMENT

System.out.println("This is my sample program");

System------> it is a predefined class[class System]

.----------> it is a dot operator,any word after it can be reference variable/object or method out----------> it is an object (predefine) println( )----> it is a method (predefine)

Parts of java :

1.J2SE/JSE (java 2 std edition) - Basic/core java

2.J2EE/JEE (java 2 enterprise edition) - Full stack development

3.J2ME/JME (java 2 micro edition)

Example

```
public class Today
{
   public static void main(String args [ ] )
   {
       System.out.println("Hi Rohan");
       System.out.print("Your order with id:1235 ");
       System.out.println("is on the way.");
       System.out.print("Please collect it");
  }
}
```

Commenting any line :

----> if we put // in starting of statement it will be commented (not considered as part of program) Ex: //public class AboutMe

```
class Today//class declaration
{
      public static void main(String ags[ ])//main method
     {
            //System.out.print("My Name is : Paul ");
            //System.out.println("I am 22 years old");
   System.out.print("I have graduated from Osmania University ");
     }
}
```

\n :

→ It is used to break the line.

→ It must be in double quotes.

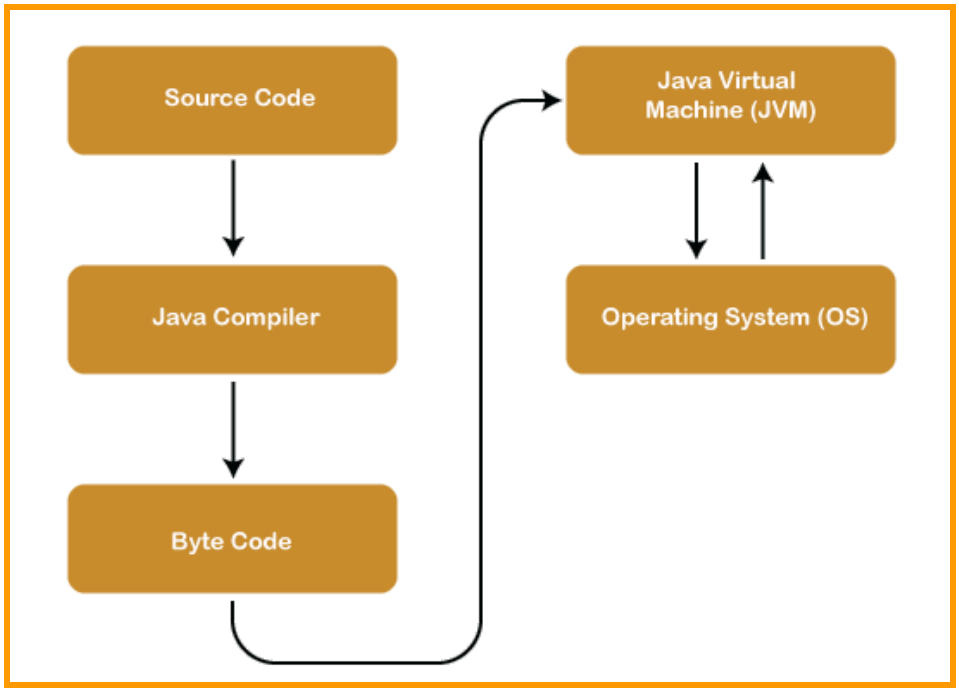
```
public class MyInfo
{
        public static void main(String args[ ])
        {
                System.out.println("Name : Rohan \n DOB : 25-07-1993 \n Age:27 ");
        }
}
```

Architecture of Java

Step-1:

-Whatever program we write is called source code.

-source code should always save as ext ".java"



Step-2:
Compilation
-The process of converting our program into system understandable form (byte code) is the purpose of compiling a program.
-to compile a program go to cmd prompt and enter command as
-javac program_name.java
-During compilation, the compiler will check syntax errors like [ ], ;,(),{ },:,spellings and case sensitivity.
-If anything is wrong we will get a compile time error.
-If nothing is wrong there is one class file get generated (byte code file) with same as .class

Step-3: Execution

-JVM -java virtual machine is responsible for execution of every java program
-JVM's can identify by ------> public static void main (String [ ] args)
-it is like one software or one program.
-execution will happen in line by line manner
-during execution JVM will find logical error of program

-for executing program go to command prompt and enter command as
-java program_name
-Once we enter this command JVM will go to class file and take first line and give to operating system for execution, once OS responds that i understood that line and it sends second line and it continues till last line like this whole code of class file gets executed.

JDK (Java Development Kit):

If we want to develop and execute a java program in our system we have to install JDK.
nb

Components of JDK
-JRE - java runtime environment
-JVM - java virtual machine
JRE: JAVA RUNTIME ENVIRONMENT

- It is used for executing java programs or running java apps.

JVM: JAVA VIRTUAL MACHINE

- It is responsible for executing every java program.

Note: when we install JDK with that JVM and JRE are available.

JIT : Just In Time compiler
- It helps JVM by taking code of class file line by line and giving it to the OS.
DATA TYPES AND VARIABLES
Data types :
1.Data
2.Data Types
3.Variables
Data :
Any information is called data.
For Ex: name,age,height,marks,percentage and salary etc
Data Type :
it defines the type of data.

Divided into 2 types:
1.Primitive(System define)
2.Non primitive(user define)

Primitive data type :

-These are system define data types
-These are fixed in there memory size
-These are 8 in numbers

Name	Size	Examples	Default values
1.byte	1 byte	10,2,5(127 is max -128 is minimum)	0
2.boolean	1 byte or no size	true or false	false
3.short	2 byte	100,220. . . .(32,767 to -32,768)	0
4.char	2 byte	A,a. . . . . empty space	
5.int	4 byte	1,2,777. . . . . .(2,147,483,647 is max)	0
6.float	4 byte	0.2,0.3,33.666.	0.0
7.long	8 byte	33333333,6565655.	0
8.double	8 byte	0.343434343,99.5555555.	0.0

Note:

-When we want 4-6 digits of accuracy we go for float else we use double
-byte,short,int,long is use for Numericals or integers
-double and float use for decimals
-when we store value greater than 2,147,438,647 we use long and we need to represent with 'l'

Non primitive data type :

-These are not fixed in their memory size.

1.String :

it is used to represent groups of char's ex: java,manual testing. . . . .

2.Arrays :

(10,20. . . .100)

Note:

-char can be used for single characters
-If we have more than one character we should use String
-Using String we can represent any type of data(info)

Variables :

-Variables are used to store the data for printing or using it in future.

1.Variable Declaration

Syntax : Datatype variablename;

Ex: int a;
float b;
char ch;
String s;

-variable name can be a combination of a-z,A-Z,0-9,$ and _

-variable name should not start with number

-whenever we declare a variable one memory block will get created .

2.Variable Initialisation

Syntax: variable_name=value;

a=100;
b=0.3333f;//mandatory to write f
ch='A';//mandatory to give ' '
s="java";//mandatory to give " "

-we can declare and initialize a variable in single statement also syntax: Datatype variable_name=value; int a=22; float b=0.2345f

Examples

1.int a=22,b=33;//valid
2.int a=33,b;//valid
3.float percentage=60.0;//invalid-----> f is not present
float b=0.334;//defaultly it will be considered as double to indicate that it is float we have to give f
4.char ch='AB';/invalid char can't be more than one character
5.float h=100f;//valid---->100.0
6.int i=0.334;//invalid---->integer can't store decimal values
7.String s="123";//valid----->System.out.println("123");
8.String d="3334+ghijk";//valid
9.double marks=100.3434d;//valid----->in double d is optional
10.long number=93939393939l;//valid----->in long l we need to keep it if we write more than 32 bit.

1Q) Can we change the syntax of the main method?

A) -JVM is responsible for executing every java program.

-JVM // the identity of jvm is public static...., if we change that then jvm will tell identity is not found

```
{
        public static void main(command line args)
}
```
-Execution of every java program will always start from main method
-Because JVM only knows main method
-If main method is not there or main method syntax is changed our program will not be executed.

2Q) Can we keep the main method private?

A) No we cannot keep the main method private because private fields cannot be accessed from outside of class.
-And JVM needs to access the main method from outside of class.
-If we keep the main method private, the program will compile but it will not be executed.
-We will get Error as Main Method not found in given class

3Q) Can we keep the main method as non-static ?

A) No, we can't write the main method as non-static because if the method is non-static, we have to create an object and JVM can't create an object on its own to access the main method.
-If we keep the main method as a non-static program it will compile but it will not be executed.

Note:
-Actually there is no Access Modifier called as default, if we did not mention public,private or protected it will be considered as default
→ which means there is no need to write like " default class Mydetails " instead we write " class Mydetails " itself as there will be default access modifier .
-Actually there is no non access modifier called as non static, if we did not mention static modifier, JVM will consider it as non static.
" public class void main (String args [ ]) " ----------> non static.

Keywords :-
- These are the reserved words or predefined words which have some reserve meaning.
- Various keywords in java are,

1.accessible keywords
public, private, protected, static, final, abstract and return. // System(not keyword) is a predefined class, it should start with a capital letter.

2.conditional keywords
if, else, else if, switch, case, break, continue, goto, const and default.

3.iterative keywords
for, while, do while

4.class level
class, package, import, extends, implements, interface, new
5.Exception level
try, catch, throw, throws, finally
6.Others
volatile, transient, synchronized, native etc…
Identifiers :-
Identifiers are the names used to identify variables, methods, classes, or other elements in the code.
- These are the names given by programmers as per convention.
Ex: class name, variable name, method name and package name.
rule ----------> should be followed
convention ----> if we don't follow also no problem

Rules for defining identifiers :

1.An identifier can be a combination of A-Z,a-z,0-9,$ and _ but standard is,
class name : starts with capital
variable name: starts with small
method name: starts with small
package name: starts with small

2. *If an identifier contains more than one word spaces are not allowed.
class My program----->Invalid
int my age---------->Invalid
public static void display details( )----->Invalid
class Myprogram----->valid
int mypercentage------>valid

3.*An identifier cannot start with a digit.
class 1A------->Invalid
int 10a---------->Invalid
class A1--------->valid
int a10-------->valid

4.class name contains more than one word for all words first letter should be capital
Ex: class MyFirstProgram

5.If a method name and variable name contains more than one word from the

second word, the first letter should be capital.
Ex: int myAge;
Ex: public static void displayDetails( )

6.*A variable name and class name cannot be a keyword.
Ex: class new----->Invalid
int static------->Invalid

OPERATORS :
Arithmetic Operators :
+ ====> Addition
- ====> Subtraction
/ ====> Division (output:Coefficient)
* ====> Multiplication
% ====> Mode (output:Remainder) =>

Java provides Unicode for every character.
A-65,B-66,C-67. . . . . .Z-90
a-97,b-98,c-99. . . . . .z-122
Ex: 'A' + 65----->65+65------>130
'a'+'z'------->97+90------>187
Ex: char ch=65;
System.out.println(ch); / / A

Concatenation
- String concatenation is the process of joining two or more strings or a string with other data types together to form a single string.
Ex: String s="java"; int a=123;
System.out.println(s+a); / / "java"+123--->"java123"
System.out.println("I am "+s+" developer"); / / "I am java"+"developer"--->I am java developer

Assignment operator(=) :
- it is used to assign or store the value into a variable.
Ex: int a = 10,b=30;
int c = a+b;
Comparison operator(==) :
- it compares values and gives output as boolean value.
Relational Operators :

- checks the relationship between two values and the result will be boolean
  > ----------> greater than
  < ----------> less than
  >= ----------> greater than equals
  <= ----------> lesser than equals
  != ----------> not equals

Result will be either true or false

Logical Operators :

AND :

- it compares two inputs and if both inputs are true then output is true or else false.
- it is represented as && (in below ex 0 indicate false, 1 indicate true)

a	b	a&&b
0	0	0
0	1	0
1	0	0
1	1	1

OR :

- it compares two inputs and if any one input is true then output is true or else false.
- it is represented as || (in below ex 0 indicate false, 1 indicate true)

a	b	a||b
0	0	0
0	1	1
1	0	1
1	1	1

Unary Operators :

a) Increment operator : ++

- It increases the value by one.

for ex: a=10; increment a add +1 to a

- It is denoted as ++
- It is of two types
  1.pre-increment
  - The rule is first increment value then print or assign it or store it in variable
  2.post-increment

- the rule is first print or assign it or store it in variable then increment value.

b) Decrement operator : --

- It decreases the value by one.

for ex: a=10; decrement a subtract -1 to a

- It is denoted as --
- It is of two types

1.pre-decrement

- the rule is first decrement value then print or assign it or store it in variable

2.post-decrement

- the rule is first print or assign it or store it in variable then decrement value.

Control flow statements

They are divided into 3 parts:

1.Conditional Statements
2. Looping Statements
3. Jump Statements

1.Conditional Statements

These statements allow you to execute certain blocks of code based on specific conditions.

Ex:

1. if
2. if-else
3. if-else if ladder
4. switch statement

2. Looping Statements

These statements allow you to repeat a block of code multiple times.

1. for loop

2. while loop

3. do while loop

3. Jump Statements

These statements control the flow of the program.

1. break
2. continue
3. return

Methods

-A set of java statements which is enclosed within curly braces having a declaration and which gets executed whenever we call it is called as method.

Syntax:Access Modifier NonAccessModifier returntypeMethodname(Arguments)

```
{
    //set of statements//
}
```

AccessModifier NonAccessModifier returntype-------->Method Header
Method_name(Arguments)--------->Method signature
{
//set of statements // -------->Method implementation or method body
}

Access Modifier

-In java we have 4 access modifiers ----------> public,private,protected and default
public static void run()
private static void run()
protected static void run()
static void run() //default

Non Access Modifiers

-In java we have 2 non access modifiers and they are static and non static.
public static void run() //static
public void run() //non static

Important Points

1. Methods are used to perform some specific task.
2. Methods should be declared within the scope of class.
3. Within one class any number of methods we can write.
4. A method will be executed only when we call it.
5. we can call a method with the help of method signature.
6. A method can be called any number of times depending on our requirement.

7. One method cannot be written in another method, it can only be called.

- The main advantage of using a method is reusability,i.e once we create a method, how many times we want we can execute it.

METHODS WITH SOME INPUT

-A method can have any number of inputs in the form of arguments.
-Ex: public static void add() //method with zero argument or 0 input
public static void add(int i) //method with integer argument
-When we call any method with an argument we have to pass that particular type of values.
-While passing arguments we have to make sure it will be as per sequence defined in method declaration.

METHOD WITH RETURN TYPE

-If we want we can define a method with any specific return type.
for ex: public static int add()

```
{ //perform operation whatever we want
  // return integer type value;
}
```

-Whenever we define a method with a return type, we are saying that my method is going to return that particular type of data(value).
-It is mandatory to write a return statement if we have a method with a specific return type.

return keyword

-It is used inside a method whenever we define a method with specific return type
-return keyword is used to take data and exit from method
-return keyword must be last statement in a method
-return keyword will not print the data.
-we cannot write more than one return statement in a method because after one return statement it exits from the method, so that the next return statement cannot be executed

TYPES OF VARIABLES

Based on the position of declaration and scope of the variables, all variables are divided into 3 types in java.

1. instance variables

2. static variables
3. local variables

1. instance variables :

- Instance variables are variables that are declared within a class but outside any methods, constructors, or blocks.
- instance variables are created (memory allocated) at the time the object is created.
- initialization of instance variables is optional, if we don't initialize then jvm takes the responsibility to initialize them with default values.
- instance variables cannot be accessed directly from static methods and static blocks.
- if we modify instance variables using one object it never impacts other objects because every object has a separate copy.
- scope and lifetime
  - the scope of instance variables is the same as object scope.
  - the lifetime of instance variables is the same as object lifetime.

2. static variables :

- Static variables are also declared directly inside the class but not inside any method or a block or a constructor but using a static keyword.
- static variables are created (memory allocated) only once at the time of classloading.
- initialization of static variables is also optional, if we don't initialize then the jvm initializes them with default values.
- scope and lifetime
  - the scope of static variables is the same as class scope.
  - the lifetime of static variables is the same as class lifetime.
- static variables can be accessed directly from static methods and static blocks.

3.local variables :

- local variables are declared directly inside a method or a block or a constructor but not inside the class directly.
- local variables are created (memory allocated) whenever method or block or constructor execution starts.
- initialization of local variables is mandatory at the time of declaration and before accessing.
- scope and lifetime
  - local variables scope is within the method or a block only.

- the lifetime of local variables is until the method or block execution completes.
- the only modifier which is allowed for local variables is final.

Is Initialization of local variables mandatory ??
Yes,it is mandatory to initialize local variables, if we did not initialize we will get compile time error saying that "Variable might not initialized"

Local variables can only be default or final i.e we cannot declare a local variable as private,public or protected.

Q. What if local variable and instance variable names are the same?
A. If local var and global var names are the same, the first priority is always given to local variables because they are inside the method and nearer for execution to JVM.

Q. Is initialization of instance variables mandatory?
A. They are not mandatory to initialize, if we did not initialize, JVM will take default values based on data types.

byte,short,int and long --------> 0
float and double ------------> 0.0
String ----------------------> null
boolean ----------------------> false
char ----------------------> empty space

LOCAL VARIABLES	GLOBAL VARIABLES
Variables which are declared inside the method and within the scope of class.	Variables which are declared outside the method and within the scope of class.
Local variables are accessible only within the method in which they are declared, they are not accessible outside of that method.	Global variables are accessible everywhere within the scope of a class.
It is mandatory to initialize local variables, if we did not initialize we will get compile time errors.	It is not mandatory to initialize, if we did not initialize, JVM will take default values based on data types.

There are no types for local variables.	Global variables are divided into 2 types i.e. static and non-static.
Local variables can be default or final.	Global variables can be public, private, protected, default or final.
Local variables are present in the Stack area.	Global variables are present in the Heap area.

METHOD OVERLOADING

The process of developing multiple methods with the same name but different arguments list is called method overloading.

RULES FOR DEFINING ARGUMENT LIST

1. Number of arguments must be different.

Ex: swiggy()
swiggy(String name)
swiggy(String name,int orderid)

2. Types(datatype) of arguments must be different.

Ex: add(int i,int j)
add(double d,double d1)
add(String s1,String s2) \

3. Sequence or position of Arguments must be different.

Ex: login(String ul,int id)
login(int id,String pwd)

-We go for method overloading when we want to perform one task in multiple ways.

Q. Can we overload main() or not ?

A. Yes, we can overload main() because JVM knows a main() which is having arguments as String args[](command line arguments), so if we define any other methods whose name is main it will be considered as user defined methods.

Note:In practical or real time it is suggested not to overload main().

Q. Can we overload final methods ?

A. Yes, we can overload final methods because the final keyword says do not change implementation and in overloading we are not changing implementation rather we are changing arguments.

Q. Can we overload private methods or not ?
A. Yes, we can overload private methods because private methods are accessible everywhere in the same class and overloading also happens within class.

Q. Can we overload non-static methods or not ?
A. Yes, we can overload non-static methods but to call them we have to create an Object.

Non static Methods
- If we do not mention the static keyword in a method it will become non-static.
  for ex: public void register() { }

Syntax for object creation

```
classname refrencevar=new classname();
                    or
classname referencevar=new constructor();
```

- new is a keyword which is responsible for creating an object.
- with the new keyword one object gets created and it loads all non static members(non static methods and non-static variables) and returns a reference address.

Conclusion points

- ➢ static variables can be accessed directly in static methods
- ➢ static variables can be accessed directly in non static methods
- ➢ Non static variables cannot be accessed directly in static methods
- ➢ Non static variables can be accessed directly in non static methods
- ➢ static method can be called directly from static methods
- ➢ static method can be called directly from non static methods
- ➢ Non-static methods cannot be called directly from static methods
- ➢ Non static method can be called directly from non static methods

EXECUTION PROCESS

Whenever we execute any program there will be two memory blocks that get created.

1. stack area also called Execution Area
2. Heap area also called as Storage Area

Important points

1. First JVM will enter into the stack area by making a call to class Loader.
2. A class loader is like a function(program) whose job is to load all static members into a static pool area(SPA).
3. SPA is a part of the heap area.
4. Once the class loader loads all static members into SPA its job is done, so it will come out of the stack area.
5. Next JVM will make a call to the main method.
6. Upon call to any method, the method will get loaded in the stack area under the method area.
7. Next Execution of the main method will start.
8. Once the entire execution is completed main() also comes out of the stack area.
9. Next JVM before coming out of the stack area it will make a call to the garbage collector whose job is to clean up the entire memory for next execution.

Feature	Stack	Heap
Storage	Stores method calls, local variables, and references to objects.	Stores actual objects and arrays.
Order	LIFO	Complex mechanism EDEN -Survivor1-Survivor2 By GC
Lifetime	Short-lived (during method execution).	Long-lived (objects remain until garbage collected).
Efficiency/Performance	Faster than Heap	Slower than stock
Allocation & De-allocation	Automatically allocated and deallocated when methods are called/returned.	Dynamically allocated and managed by the Garbage Collector.

Differences Between Static Pool Area and Metaspace

Feature	Static Pool Area	Metaspace
Purpose	Stores **constants** and references to **static data** (e.g., string literals, static variables).	Stores **class metadata** (information about loaded classes).
Part of	Part of the **Method Area** (pre-Java 8).	Part of **native memory** (replaces PermGen from Java 8 onwards).
Memory Location	Stored in **JVM heap** (string pool in Java 8+).	Stored in **native memory** outside the Java heap.
Memory Management	Static Pool is part of the **JVM heap memory** and can be managed by garbage collection for things like string literals.	**Metaspace** grows dynamically and is managed by the JVM.
Lifetime	Lifetime tied to the class loading and the constant pool.	Lifetime tied to the class loader and class unloading.
Errors	Can lead to memory issues related to constant pool, but usually, it's managed well by the JVM.	If Metaspace exceeds allocated memory, it can cause an `OutOfMemoryError`.

What is the String Constant Pool?

- The String Constant Pool is a special memory area where String objects are stored.

Static	Non static
1.static means single copy.	1.Non static means multiple copies
2.static members are present in the static pool area.	2.Non static members are present in the object.
3.static members can be accessed in 3 ways 1.Directly 2.Through class name 3.Through object reference	3.Non static members can be accessed through object
4.static members will get loaded only once in SPA	4.Non static members will get loaded whenever we created an object

5.class loader is responsible for loading static members.	5.new keyword is responsible to load all non static members
6.Any changes made to static variables will affect the entire class that is why static variables are also called as class variables.	6.Any changes made to non-static variables will affect only a particular object that is why they are called as instance variables.
7.We go for static, if the data is fixed. Static String clg=”Qspiders”;	7.We go for non static, if the data is changing. String courses=”CSE”;
8.static methods cannot be inherited.	8.Non static methods can be inherited.
9.static methods cannot be overridden.	9.Non static methods can be overridden. 07

SCANNER CLASS

• Scanner is a predefined class(already defined) which is used to take an input from user during runtime or execution time.

• This Scanner class is used when we want to take inputs from user not from programmer.

Steps to use Scanner class

- -

1. Write first statement in program as

```
import java.util.Scanner;
        OR
import java.util.*;
```

import: it is a keyword which indicates that we are using any other class code in our own class.

2. Create an Object of Scanner class

```
Scanner s=new Scanner(System.in);
```

- here System.in indicates that we are reading input from the system.

3. Using Scanner class reference call Scanner class methods

methods of Scanner class are:
nextByte() ----------> For taking byte value as input
nextBoolean() --------> For taking boolean value as input
nextShort() ----------> For taking Short value as input
nextInt() ----------> For taking integer value as input
nextFloat() ----------> For taking float value as input
nextDouble() ----------> For taking double value as input
nextLong() ----------> For taking long value as input
next() ----------> For taking String value as input without space
nextLine() ----------> For taking String value as input with space

Constructor

Constructor is a special type of method which gets executed whenever we created an object

- The main purpose of constructor is to initialize a non static variable.

Syntax: AccessModifier consructo_name(arg/no args)

```
{
        //body of constructor
}
```

Rules for defining constructor

1. Constructors can be public, private, protected or default.
2. Constructors cannot be static, non static, final or abstract.
3. Constructor name must be the same as that of classname.
4. Constructor does not have any return type, not even void.
5. Constructors can be with arguments or without arguments.

Types of Constructors:-

They are of 2 types-
1. No argument Constructor
2. Parameterized Constructor

No argument Constructor

-If a constructor does not have any argument it is called a No argument constructor.

```
class Student                    {
        String name;
        public Student()
        {
           name="John";
        }
        public static void main(String args[])
        {
                Student s1=new Student();
                System.out.println(s1.name);
                Student s2=new Student();
                System.out.println(s2.name);
                Student s3=new Student();
                System.out.println(s3.name);
        }
}
```

One of the drawbacks of no argument constructor is, it provides the same value for every object.

- Therefore to overcome this drawback, we go for parameterized constructor.

Parameterised Constructor

- If a constructor contains any arguments, it is called as parameterised constructor.
- While creating Object, we have to pass particular argument as constructor.

Examples:

```
class Student
{
        String name;
        int age;
        public Student(String sname,int sage)
        {
                name=sname; age=sage;
        }
        public static void main(String args[])
        {
                Student s1=new Student("John",22);
                System.out.println(s1.name+" "+s1.age);
```

```
        Student s2=new Student("Rohan",24);
        System.out.println(s2.name+" "+s2.age);
        Student s3=new Student("Rahul",24);
        System.out.println(s3.name+" "+s3.age);
    }
}
```

METHODS	CONSTRUCTORS
1. Since, we can't write business logic directly in class, we use methods.	1. Constructors are mainly used for intialising Non static variables.

2. Syntax:
AccessModifier NonAccessModifier reurntype methodname(args/no args)
{
}
 	
	2. Syntax:
	AccessModifier Constrctorname(args/no args)
	{
	}

3. Method name can be anything(as per user).	3. Constructor name must be same as classname.
4. To execute a method, we have to call it static------->Directly Non static--->Object	4. Constructor will get called automatically, whenever we created an Object.
5. We can call one method from another method directly or through object.	5. We can call one constructor from another constructor through consructor chaining.
6. Only Non static methods can be inherited.	6. Constructors cannot be inherited.
7. Only Non static methods can be Overridden.	7. Constructors cannot be Overridden.

OOPS

1. Class
2. Object
3. Inheritance
4. Abstraction
5. Encapsulation
6. Polymorphism

Class and Object

Aspect	Class	Object
Definition	A blueprint or template for creating objects.	An instance of a class, created using the class blueprint.
Memory Allocation	No memory is allocated until objects are created.	Memory is allocated when an object is created.
State/Behavior	Defines properties and behaviors.	Holds specific data (state) and can perform behaviors (methods).
Creation	Defined once using `class` keyword.	Created using `new` keyword and a constructor.
Example	`class Car { ... }`	`Car myCar = new Car();`

Inheritance

- Acquiring the properties of one class into another class is called inheritance.
- to use inheritance we use extends keyword.
- inheritance establishes is-a (parent-child) relationship between the classes.

Types of Inheritances

1.Single Level
2.Multi Level
3.Hierarchical
4.Multiple
5.Hybrid

1. Single Level Inheritance

definition :- One superclass and One sub class

```
class Bank
{
	String accname="John";
	int accno=245678;
	double avalbal=100;
	public void details()
	{
		System.out.println("Account Holder : "+accname+" accno:"+accno);
	}
}
class Deposit extends Bank
{
```

```
        double amt=4550.5;
        public void depositamt()
        {
                avalbal=avalbal+amt;
                System.out.println("Total balance after depositing Amt :"+avalbal);
        }
}
public class Cust
{
        public static void main(String args[])
        {
                Deposit d1=new Deposit();
                d1.details();
                d1.depositamt();
        }
}
```

2. Multi-level Inheritance

definition :- Two super classes and Two sub classes

```
class Bank
{
        int accno=23456;
        String accname="John";
        double availbal=2000;
        public void details()
        {
                System.out.println("Acc name : "+accname+" Acc no : "+accno);
        }
}
class Deposit extends Bank
{
        double amt=6000.5;
        public void deposit()
        {
                availbal = availbal + amt;
                System.out.println("Amount deposited : "+availbal);
        }
}
class Withdrawl extends Deposit
{
```

```
	double wamt=3000;
	public void witamt()
	{
		availbal = availbal - wamt;
		System.out.println("Withdrawl amount : "+availbal);
	}
}
public class Transaction
{
	public static void main(String args[])
	{
		Withdrawl w1 = new Withdrawl();
		w1.details();
		w1.deposit();
		w1.witamt();
	}
}
```

3. Hierarchical Inheritance

definition :- One superclass and Two sub classes

```
class Bank
{
	int accno=23456;
	String accname="John";
	double availbal=2000;
	public void details()
	{
		System.out.println("Acc name : "+accname+" Acc no : "+accno);
	}
}
class Deposit extends Bank
{
	double amt=6000.5;
	public void deposit()
	{
		availbal = availbal + amt;
		System.out.println("Amount deposited : "+availbal);
	}
}
```

```
class Withdrawl extends Bank
{
        double wamt=3000;
        public void witamt()
        {
                availbal = availbal - wamt;
                System.out.println("Withdrawl amount : "+availbal);
        }
}
public class Transaction
{
        public static void main(String args[])
        {
                Deposit d1 = new Deposit();
                d1.details();
                d1.deposit();
                Withdrawl w1 = new Withdrawl();
                w1.details();
                w1.witamt();
        }
}
```

4. Multiple Inheritance

- Multiple inheritance is One class is inheriting two immediate super classes at the same time
- But in java, a class can extends only one class at a time
- So Multiple inheritance is not possible through classes because of Ambiguity problem and Constructor chaining problem.
- If one class extends two classes and in case, if both classes contains same method then while calling a method, JVM will get confuse which class method to call this problem is known as Ambiguity problem.

Q> Can we inherit Constructors or not?

A. • No we cannot inherit constructors because they are not member of a class(members of classes are methods and variables) and constructors are mainly used for initialization of Non-static variable.

- Constructors cannot be inherited but they can be invoked by using call to super.

Call to super

- The process of calling one constructor from another constructor of different class

is called as call to super.

• Call to super must be the first statement in constructor.

```
class A
{
        public A(int i)
        {
                System.out.println("A class default constructor");
        }
}
class B extends A
{
        public B()
        {
                super(100);
                System.out.println("B class Default Constructor");
         }
}
public class Check
{
        public static void main(String args[])
        {
                B b1=new B();
        }
}
```

Constructor chaining problem

```
class A
{
         public A()
        {
         }
}
class B
{
        public B()
        {
        }
}
class C extends A,B
{
```

```
        public C()
        {
                super();//Constructor chaining problem
        }
}
```

• In the above program, JVM will get confused which super class constructor to call because there are two super classes. Hence, it gets confused this problem is known as Constructor chaining problem
• Therefore due to ambiguity and constructor chaining problems.
Multiple inheritance is not possible through class but possible through Interface.

Hybrid Inheritance

• It is a combination of Multiple and Hierarchical Inheritance since Multiple is not possible through classes. Hybrid is also not possible through classes.

Advantages of Inheritance :

1. Reusability of code
2. Avoid duplicacy of code

Note :

To every class there is a default superclass present, whether we write it or do not write it. i.e, Object class.
• Object class contains 9 methods and these methods are used in multiple classes of java. So, instead of defining it separately in all classes they have defined it in object class and make it as super class. So that without defining it again and again other classes can simply use it.

Note:

1. We can inherit public,protected and default methods.
2. We cannot inherit private methods because they are accessible only within class.
3. We can inherit non static, final and abstract methods.
4. We cannot inherit static methods because they will be loaded only once in SPA.
5. We cannot inherit constructors because they are mainly used for initialisation and they are not members of class.

HAS-A relationship

• One class containing the reference of another class is called a HAS-A

relationship.

```
class Engine
{
        //properties of engine
} class Car
{
        Engine e = new Engine();//Car has-a reference of Engine
 }
```

Method Overriding

During Inheritance subclass has complete privilege to change the (method) implementation of super class, this process is known as Method Overriding.

```
        class Parents
        {
                public void car()//Overridden method
                {
                        System.out.println("Blue color");
                }
                public void carname()
                {
                        System.out.println("Audi");
                }
        }
        class Son extends Parents
        {
                public void car()//Overriding method
                {
                        System.out.println("Black color");
                }
        }
        class Daughter extends Parents
        {
                public void car()
                {
                        System.out.println("Pink color");
                }
                public void carname()
                {
```

```
                System.out.println("Nano");
        }
}
public class Driver
{
        public static void main(String args[])
        {
                Parents p1 = new Parents();
                p1.car();
                p1.carname();
                Son s1 = new Son();
                s1.car(); s1.carname();
                Daughter d1 = new Daughter();
                d1.car();
                d1.carname();
        }
}
```

Rules For Overriding

1. Inheritance is compulsory.
2. Method signature must be the same (methodname and arguments)as superclass in subclass.
3. method header must be the same (access modifier, non access modifier, return type)
4. Overridden method should not be final.

Polymorphism

• It is a Greek word.
• poly means many and morphism means forms.

One thing showing multiple behaviour is called polymorphism.

(or)

One entity showing different behaviour is called polymorphism.

Types Of Polymorphism :

1. compile time or static polymorphism

- the polymorphism which is shown at compilation time by the compiler is called compile time or static polymorphism.

ex: method overloading, method hiding

2. runtime or dynamic polymorphism

--> the polymorphism which is shown at runtime by the jvm is called as runtime or dynamic polymorphism.

ex: method overriding

Encapsulation

• The process of wrapping the data members(Global variables) and member function(methods) into a single unit(class) is called Encapsulation.
• The main purpose of encapsulation is to achieve security of data.

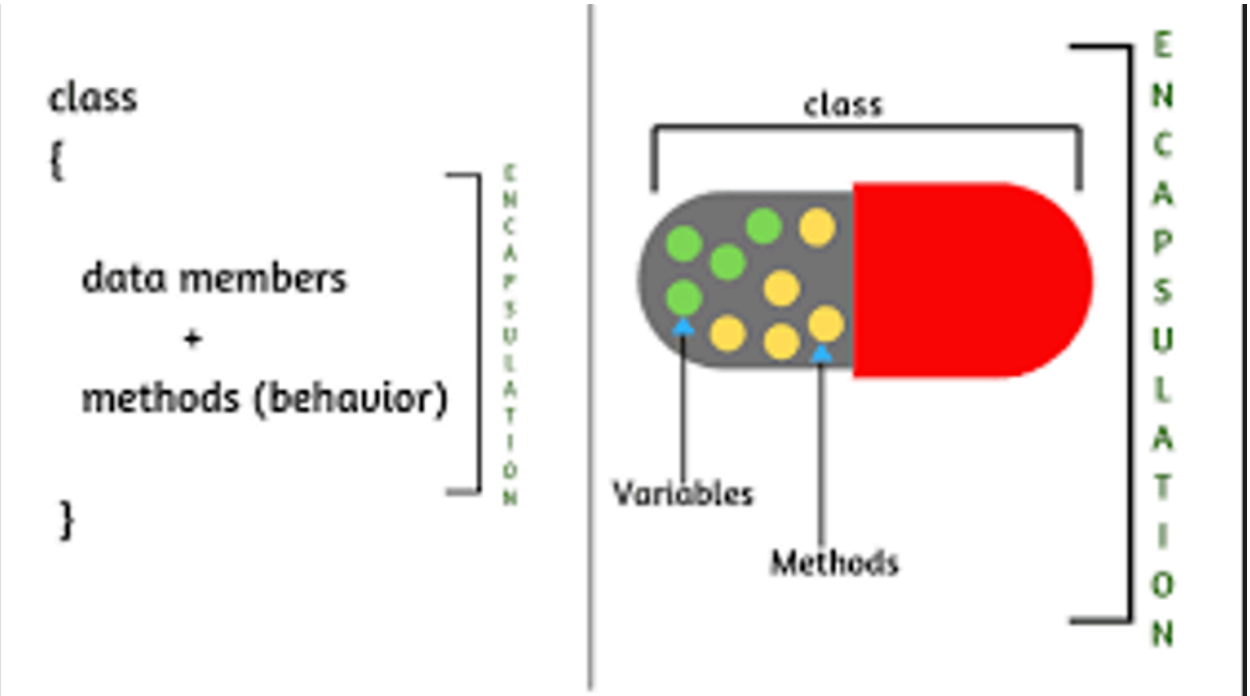


Fig: Encapsulation

What is the need of Encapsulation ?

```
public class demo
{
        public static void main(String args[])
        {
                App a1 = new App();
```

```
            a1.monthnum = 22;
            System.out.println("Month number is "+a1.monthnum);
        }
}
//Unencapsulated class
class App {
        int monthnum;//datamember is not protected by developer or programmer
}
```

• We go for encapsulation to protect our data members from invalid users.
• For ex : if monthnum is not protected(Encapsulated) user may misuse the datamember by assigning invalid values(like >12 or <1)
• If a data member is not protected(public), it is under user control i.e, the user will decide what values to assign but those values can be valid or invalid.
• If a data member is protected(private), it is under developers control to decide which values to be allowed.

Purpose

• Protecting the data members by keeping them private and accessing through some special methods is nothing but Encapsulation.

Rules for Encapsulation :

1. Declare all the data members as private (if a datamember is private, it is not accessible outside of class).
2. define separate setter and getter methods (it's not mandatory to define only setter and getter, we can define any user defined methods).

```
public class UserCal
{
        public static void main(String args[])
        {
            Calendar c1 = new Calendar();
            c1.monthnum=22;
            System.out.println("Month number is "+c1.monthnum);
        }
}
//Encapsulated class
class Calendar
{
        private int monthnum;//data member is protected by developer or
        programmer
}
```

```
class Login //ex-using Encapsulation
{
	private String username;
	private String pwd;
	public void setusername(String username)
	{
		this.username=username;
	}
	public String getusername()
	{
		if(username=="John")
		{
			return "Username is correct, Please Enter Password" ;
		}
		else
		{
			 return "Username is Incorrect" ;
		}
	}
	 public void setpwd(String pwd)
	{
		this.pwd=pwd;
	}
	public String getpwd()
	{
		 if(pwd=="John@14141")
		{
			return "Please go ahead" ;
		}
		 else
		{
			return"Entered password is invalid" ;
		}
	}
}
public class PageUser
{
	public static void main(String args[])
	{
		Login l1 = new Login();
```

```
            l1.setusername("John");
            System.out.println(l1.getusername());
            l1.setpwd("Something@143");
            System.out.println(l1.getpwd());
        }
}
```

JAVA BEAN CLASS :
A class is called a Java Bean class, if all data members are private and there is a separate setter and getter method for each data member.

setter method
1. It is public in nature.
2. The setter method does not have a specific return type.
3. The setter method contains arguments the same as data members.
4. The name of the method is set followed by the data member name.

getter method
1. It is public in nature.
2. The return type must be the same as that of the data member.
3. The name of the method is followed by the data member name.
4. The getter method does not have arguments.

Note : • For every data member we have to define a separate setter and getter method.

```
import java.util.Scanner; //EX
class EPF
{
        private long eidPF;
        private String pwd; //setter method---eidPF
        public void seteidPF(long eidPF)
        {
             this.eidPF=eidPF;
        }
        //getter method---eidPF
         public long geteidPF()
        {
            if(eidPF==1011234581)
            {
                System.out.println("Processing your EID please wait.....");
```

```
                return eidPF;
            }
            else
            {
                System.out.println("Incorrect EID"); return 0;
            }
        }
        //setter method---pwd
        public void setpwd(String pwd)
        {
            this.pwd=pwd;
        }
        //getter method---pwd
        public String getpwd()
        {
            if(pwd=="Ravi123")
            {
                System.out.println("Processing your password please
                wait.....");
                return "Password is Correct";
            }
            else
            {
                System.out.println("Incorrect Password");
                return "Please check your Passsword again";
            }
        }
}
public class EmployeeEP
{
        public static void main(String args[])
        {
            Scanner sc = new Scanner(System.in);
            EPF e1 = new EPF();
            System.out.println("Please Enter your EmployeePF-EID");
            long eid = sc.nextLong();
            e1.seteidPF(eid);
            System.out.println(e1.geteidPF());
            System.out.println("Please Enter your Password");
            String pwd = sc.nextLine();
```

```
            e1.setpwd(pwd);
            System.out.println(e1.getpwd());
        }
}
```

Q. System and Scanner are predefined classes but why do we write import statements for Scanner class only ??

Classes present in java.util package can only be used by writing import statements, whereas classes present in java.lang package can be used directly.

Fully Encapsulated class

• If all data members of a class are private and there are separate methods to access those data, such a class is called a Fully Encapsulated class.

```
class A
{
        private int i;
        private float f;
        private String s; //methods to access them//
}
```

Partially Encapsulated class

• if any data members of a class is non private, such class is called a Partially Encapsulated class.

```
class A {
        private String name;
        public int age;
}
```

No Encapsulated class

• if none of the data members of a class is private, such class is called a No Encapsulated class.

```
class A
{
        public String name;
        public int age;
}
```

Advantages Of Encapsulated class :

• Securing the data
• We can make our class write only by writing only the setter method.
• We can make our class read only by defining only the getter method.

Dis-advantages Of Encapsulated class :

• length of the code increases.

Object Casting

• Converting one type of object into another type of object is called Object Casting.
They are of 2 types :-
1.Upcasting
2.Downcasting

Upcasting

• Converting sub class object type into super class object type is called upcasting.
(or) • creating an object of subclass and store it into a reference of super class is called upcasting.
• During up-casting only superclass behaviour is visible, sub class behaviour is hidden.

Downcasting

• Converting a super class object type into a subclass object type is called downcasting. (or)
• Converting an upcasted object into normal form is called downcasting.
• During downcasting both subclass and superclass behaviour is visible.
• Since, superclass does not contain property of sub class directly, downcasting is not possible explicitly we have to convert it.

Note:during upcasting some properties are hidden, during downcasting all properties are visible.

Type(datatype) Casting

• Converting one type of primitive data type into another type of primitive data type is called type casting.

(Or)

• Assigning one type of value into another type is called Type casting.
• It is divided into two types -
1.Widening
2.Narrowing

Widening

• Converting smaller primitive data types into bigger primitive data types is called widening.
• Widening is also called Implicit Casting.
• byte-->short-->int-->long-->float-->double
• Since, we are converting a smaller type into a bigger type, there is no loss of data.

```
public class Widening //Ex
{
        public static void main(String args[])
```

```
    {
        byte b=45; //widening
        short s=b;
        int i=s;
        long l=i;
        float f=l;
        double d=f;
        System.out.println(s);
        System.out.println(i);
        System.out.println(l);
        System.out.println(f);
        System.out.println(d);
    }
}
```

Narrowing

• Converting a bigger primitive data type into a smaller primitive data type is called Narrowing.
• Narrowing is also called Explicit Casting.
• byte<--short<--int<--long<--float<--double
• Since, we are converting a bigger type into a smaller type, there is loss of data.

```
public class Narrowing //Ex
{
    public static void main(String args[])
    {
        double d=163.45;
        float f=(float) d;
        long l=(long) f;
        int i=(int) l;
        short s=(short) i;
        System.out.println(d);
        System.out.println(f);
        System.out.println(l);
        System.out.println(i);
        System.out.println(s);
    }
}
```

Abstraction

• The process of hiding the internal implementation and showing the necessary data to the end user is called as Abstraction.

Examples:
Driver knows only to apply brakes and accelerate, but how it is working is hidden.
In JAVA abstraction can be achieved in two ways :-
1. Using Abstract class
2. Using Interface

Abstract class

• abstract is a keyword, which indicates incompleteness.
• abstract class is a class which contains at least one abstract method.

```
abstract class Vehicle //Ex
{
        abstract public void noOfWheels();
}
```

Normal/Concrete/Complete method

• If a method has method signature as well as method body, such method is called Complete or Concrete method.

```
Ex: public void run()
{
SOP("In run");
}
public void fly()
{
SOP("fly away");
}
```

Abstract/Incomplete method

• If a method contains only method signature but not method implementation is called an abstract or incomplete method.
Ex:
public void run()
public void fly()
• Abstract method has to be represented with abstract keyword and we have to add ; in the end of method declaration.
Ex:
abstract public void run();
abstract public void fly();
• If we don't add abstract keywords or semicolons, we will get compile time errors

• It is mandatory to mention the abstract keyword for abstract methods and abstract classes.

```
abstract class Fruit //Ex-1
{
        abstract public void taste();
}
abstract class Loans //Ex-2
{
        abstract public void type();
        abstract public void rateOfInterest();
}
```

Q. Can we create an object for abstract class?

A. No, we cannot create an object of an abstract class because it contains abstract methods and abstract methods do not have anybody to execute.

```
abstract class Vehicles //Ex
{
        abstract public void noOfWheels();
}
 class A
{
        Vehicles v1=new vehicles();//instantiation is not possible
        v1.noOfWheels();
 }
```

Q. Can we Inherit abstract classes or not ?

A. Yes, if any child classes are there for abstract class that child class has to undergo any of the one rule.

1. Complete all incomplete methods of abstract class by means of overriding.
2. Declare your class also as an abstract class.

Note : • Abstract classes can have complete as well as incomplete methods.
• Abstract classes can have static, non static and final variables.

Q. Can we define Constructor in abstract class ?

A. Yes, we can define a constructor in abstract class.

```
abstract class Student
{
        String sname;
        int sid;
        public Student(String sname,int sid)
        {
```

```
        this.sname=sname; this.sid=sid;
    }
    abstract public void details();
}
class Admin extends Student
{
    public Admin()
    {
        super("John",1234);
    }
    public void details()
    {
         System.out.println(sname+" "+sid);
    }
}
public class User1
{
    public static void main(String[] args)
    {
        Admin a1 = new Admin();
        a1.details();
    }
}
```

Q. Can we declare the class as an abstract class even though it does not contain abstract methods ?

A. Yes, we can create the class as abstract even though it doesn't contain abstract methods. In 2 cases,

• Case 1: If all the members of class are static members then to access them, we do not have to create an object we can call directly or through class name.

```
abstract class A
{
    static int i=100; main()
    {
        SOP("In main);
        SOP(i);
        run();
    }
    public static void run()
    {
        SOP("In run");
```

```
    }
}
```
• Case 2: In general, we never create an object of superclass, we can declare super class as abstract class.
```
abstract class A
{
}
class B extends A
{
}
```
Q. Can we declare an abstract class as final ?
A. No, we cannot declare an abstract class as final because if it is final it cannot be extended or inherited.
Q. Can we declare an abstract method as final ?
A. No, we cannot declare an abstract method as final because if it is final it cannot be overridden and we must have to override abstract method to complete it.
Q. Can we declare abstract methods as static ?
A. No, we cannot declare an abstract method as static because if it is static it cannot be overridden and we must have to override abstract method to complete it.
Q. Can we achieve Multiple Inheritance through abstract class ?
A. No, we cannot achieve Multiple Inheritance through abstract class because one class cannot extend more than one abstract class.
Conclusion:
• Since, Abstract class contains complete as well as incomplete methods. We are able to achieve partial abstraction i.e. 0 to 100% abstraction. Therefore, to achieve 100% abstraction, we go for an interface.

Interface:
- An **interface** in Java is a collection of abstract methods and constants that a class can implement.
- It provides a way to achieve abstraction and multiple inheritance of method signatures in Java.
```
interface Animal
{
        void eat();
        void makenoise();
        void color();
        void species();
}
```
• An interface has to be represented with an interface keyword.

• Syntax: interface_keyword interface_name
{
//body of interface
}
• By default all the methods of interface are public and abstract whether we write or don't write.
• By default all the variables of interface are public, static and final whether you write or you don't write it.
Q. Can we instantiate an interface ? (or) Can we create an object of interface ? A. No, we cannot create an object of interface because all methods are by default abstract. But we can create a reference to the interface.

Implements
• implements is the keyword, we will use if any class wants to form a relationship with interface.
syntax:

```
interface Car
{
        void price();
}
class Audi implements Car
{
}
```

Ex

```
public interface Employer
{
        void joiningProcess();
        void pF();
        void allocateWork();
}
public class Employee
{
        public static void main(String[] args)
        {
                HrDeptInf i1=new HrDeptInf();
                i1.joiningProcess();
                i1.allocateWork();
                i1.pF();
                i1.location();
        }
```

```
}
 abstract class Infosys implements Employer
{
	@Override
	public void joiningProcess()
	{
		System.out.println("1.Selection \n2.Document Verification\n3.Send
		Offer Letter");
	}
	@Override
	public void pF()
	{
		 System.out.println("As per the standard norm of EP");
	}
	abstract public void location();
}
class HrDeptInf extends Infosys
{
	@Override
	public void allocateWork()
	{
		 System.out.println("Cycle-1 is for Development\nCycle-2 is for
		QA");
	}
	@Override
	public void location()
	{
		System.out.println("For training-Bangalore Deployment-Hyderabad");
	}
}
```

Multiple Inheritance through Interface

```
interface A
{
	void m1();
}
interface B
{
	void m2();
```

```
}
class C implements A,B
{
	public void m1()
	{
		System.out.println("In m1 method");
	}
	public void m2()
	{
		System.out.println("In m2 method");
	}
}
public class Main
{
	public static void main(String[] args)
	{
		C c1=new C();
		c1.m1();
		c1.m2();
	}
}
```

Q. Why is Multiple inheritance possible through interface?

• A class cannot extend more than one class but it can implement multiple interfaces.

• In interface there is no possibility of Ambiguity problem because even though multiple interfaces contain the same method name, but implementation will be given only once because an interface class gives implementation and it won't implement multiple methods with the same name and same argument in a single class.

```
interface A //Ex
{
	void m1();
}
interface B
{
	void m1();
}
class C implements A,B
{
```

```
        public void m1()
        {
                System.out.println("In m1 method");
        }
}
public class Main
{
        public static void main(String[] args)
        {
                C c1=new C(); c1.m1();
        }
}
```

• Since interface does not contain Constructors there is no possibility of Constructor chaining in interface.
• Therefore, when there is no ambiguity problem and constructor chaining problem, we can achieve multiple inheritance through interface.

Advantages :

1. We can achieve 100% abstraction.
2. It makes complex designs into simple forms.
3. Achieve multiple inheritance.

Difference Between Abstract class and Interface

abstract class A
{
}
 	
	interface A
	{
	}

class keyword must be preceeded with abstract keyword.	interfacename must be preceeded with interface keyword.
abstract class can have complete as well as incomplete methods.	Interface methods are by default abstract.
In abstract class we can hv public,private,protected,static,non static,final and non final variables.	In interface all variables are by default static, public and final.
Constructors are allowed in abstract class.	Constructors are not allowed.
Using abstract class, we can achieve 0-100 % abstraction.	Using interface, we can achieve 100% abstraction.
A class can extend only one abstract class at a time. abstract class A{} abstract class B{} class C extends (either A or B)	A class can implement multiple interfaces at a time. interface A{} interface B{} class C implements A,B
Using Abstract class, we cannot achieve multiple inheritance.	Using interface, we can achievemultiple inheritance.
We go for abstract classes, when we knows partial implementation.	We go for interface when we only knows the specification but not implementation.

Ex:
abstract class EMP
{
public void getsal()
{
SOP("CAL per Annum");
}
abstrct p v getincentives();
}
	Ex:
	interface EMP
	{
	void getsal();
	void getincentives();
	}

Final Keyword

• Final is a keyword which basically indicates that no more changes are allowed.
• final keyword is applicable with -
1.methods
2.variables(Local and Global)
3.class

** Difference between static and final

static	final
static means single copy.	final means fixed copy.
it is not mandatory to intialise static while declaration.	it is mandatory to intialise final while declaration.
static variable can be reintialised.	final variable cannot be reintialised.
A class cannot be static.	A class can be final.
static keyword is applicable with method, variables, blocks and nested class.	final is applicable with methods, variables and classes.
Ex:static String clg="Qspiders";	Ex:final float pie=3.414f;

final with class

• if a class is declared as final, we cannot inherit or extend that class. So,therefore a super class can never be final.

final with methods

• if a method is declared as final we cannot override it.
• The final keyword says that do not make further changes and in overriding we are changing the implementation. Therefore, final methods cannot be overridden.

Q. Will final methods get inherited ?

Yes, final methods will get inherited but the only thing is they cannot be overridden.

Q. Can a constructor be final ?

No, a constructor cannot be final because the final keyword is applicable only for class, methods and final not with constructors.

super keyword

• it is used to call super class instance members(non static variables and non static methods).
• it is used in sub class non static methods.

Differences between :-

this keyword	super keyword
this keyword indicates the current object under execution.	super keyword is used to call super class instance members.
this can be used inside methods and constructors.	super is used in subclass non static methods.
For using this keyword inheritance is not required.	For using super inheritance is required.

Access Specifiers :-

• It provides accessible permission to various fields of program.
• basically they are of 4 types -
1.public
2.private
3.protected
4.default
• They are applicable to
1.class
2.method(static & non static)
3.variables(static & non static)
4.constructors
5.Interface

package:

It is like a folder where all common classes kept together at one place • In java we have predefined packages as well as user defined packages.
• Some examples of predefined packages are -
1.java.lang package
2.java.util package

1.public specifier :-

• it is the lowest level specifier.
• public fields are accessible within class anywhere.
• public fields are accessible within another class anywhere of the same package.
• public fields are accessible within another class of another package only after import statements.

2.default specifier :-

• In java there is no such word called as default i.e if we did not specify any modifier like public, private or protected. JVM will consider it as default.
• default fields can be accessible anywhere within class.
• default fields are accessible outside of class but that class should belong to the same package.
• default classes cannot be imported.
• The default specifier is also called as package specifier because it is restricted within the package.

3.private specifier :-
• it provides the highest level of restrictions.
• private members can only be accessed within the same class.
• private fields cannot be accessed outside of declared class.
• A class can never be private.
• private methods can't be inherited and overridden.

4.protected specifier :
• protected fields can be accessible anywhere within class.
• protected fields can be accessible within another class but that class should belong to the same package.

Object class :
• it is super class to all pre define and user defined classes.
• Object class is present in the java.lang package.
• javap java.lang.Object
Method of object class:
1. getclass():class----->final
2. object1.equals(object2):boolean
3. toString():String
4. hashcode():int
5. notify():void----->final
6. notifyAll():void---->final
7. wait():void------->final
8. wait(long timeout):void---->final
9. wait(long timeout,int nanos):void---->final
10.finalise()
11.clone()
toString():
• It's a method of object class.
• When we call toString()on any object, it provides complete information about an

object.
• Complete information consists of packagename.classname@objectaddress
Ex:mypackage.Sample@zzh33453
Syntax:
public String toString()
{
//return packagename.classname@objectaddress;
}
• Object address is a unique address given to every object.
• it cannot be the same for 2 objects.
• Whenever we print a reference variable implicitly, it calls toString().

```
public class Sample //public class Sample extends Object
{
	public static void main(String[] args)
	{
		Sample s1=new Sample();
		System.out.println(s1.toString());
		System.out.println(s1.hashCode());
		Sample s2=new Sample();
		System.out.println(s2.toString());
		System.out.println(s2.hashCode());
		Sample s3=s1;
		System.out.println(s3.toString());
		System.out.println(s3.hashCode());
		System.out.println(s1);//s1.toString()
		System.out.println(s2);//s2.toString()
		System.out.println(s3);//s3.toString()
	}
}
```

hashcode()
• It's a method of object class.
• Whenever we call hashcode() on any object, it prints the hashcode number for a given object.
• Hashcode number is simply a 32 bit integer number.
• it is a unique number allocated to every object by JVM.
• if the object addresses are the same they will have the same hashcode number.

Overriding of toString() and hashCode()

```
package objectdemo.in;
public class Student
{
	String sname;
	int sid;
	public Student(String sname,int sid)
	{
		this.sname=sname;
		this.sid=sid;
	}
	public String toString()
	{
		return sname;
	}
	 public int hashCode()
	{
		return sid;
	}
	public static void main(String[] args)
	{
		Student s1=new Student("John",1221);
		System.out.println(s1.toString());
		System.out.println(s1.hashCode());
		Student s2=new Student("rohan",1222);
		System.out.println(s2.toString());
		System.out.println(s2.hashCode());
	}
}
```

equals()

- It's a method of object class that compares two objects based on object address.
- if the object address is the same, output is true else it is false.

```
 public class Sampl
{
	//Ex public static void main(String[] args)
	{
		Sampl s1=new Sampl();
		Sampl s2=new Sampl();
```

```
        Sampl s3=s1;
        Sampl s4=s2;
        System.out.println(s1+" "+s2+" "+s3+" "+s4);
        System.out.println(s1.equals(s2));
        System.out.println(s2.equals(s3));
        System.out.println(s1.equals(s3));
        System.out.println(s4.equals(s2));
    }
}
```

ARRAYS

• An array is a collection of homogeneous elements(data).
• Whenever we want to use multiple or groups of elements or data at same time we go for arrays.
• For example: if I want to use 1 to 100 at the same time, it's not possible to store in variables and use it because it is a very lengthy process so for such kind of situation, java has given arrays.
• In java an array is an Object.

Syntax:

```
arraytype arrayname[]=new arraytype[size];
arraytype[] arrayname=new arraytype[size];
arraytype []arrayname=new arraytype[size];
```

Ex:

```
int a[]=new int[3];
double []d=new double[3];
float[] f=new float[4];
```

• array stores the elements in index format which always starts from 0(for above example it is 0,1,2).
• Once an array gets created there will be default values stored in that array object.
• Once we create an array default values will be present.
• storing elements in an array by using index values

```
a[0]=10;
a[1]=20;
a[2]=30;
doubled[]=new double[4];
d[0]=0.22;
d[1]=0.33;
d[2]=0.44;
```

d[3]=0.55;

• for printing elements of array

System.out.println(a[0]);
System.out.println(a[1]);
System.out.println(d[0]);
System.out.println(d[1]);
System.out.println(d[2]);

• When we already know elements of array we can also create the array as

arraytype arrayname[]=(elements);
int a[]={10,22,33,44,555,666,7777,88);
System.out.println(a[0]);
System.out.println(a[1]);
System.out.println(a[2]);
|
|
So on.....

• if we store elements more than declared size we will get array index out of bounds exception.

ex:

int a[]=new int[3];
a[0]=11;
a[1]=22;
a[2]=33;
a[3]=44;//array index out of bounds exception// Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:4

Length variable:

• It provides length of array and length will always calculated from

1. ex: int a[]=new int[300];
System.out.println(a.length);
Output is :300

• Size of the array cannot be a decimal value.

int a[]=new int[4.0];//CTE

• it is mandatory to give size of array if we did not given size, we will get CTE int a[]=new int[];//CTE

• Array stores only homogeneous data i.e, in integer array we can add only integer data, if we add any other type of data we will get compile time error.

EXCEPTION HANDLING

- Exception
- Hierarchy of Exception
- Types of Exception(Different classes in Exception)
- Keywords of Exception try, catch, throw, throws and finally
- difference between throw and throws
- **difference between final, finally, finalise
- ***user defined exception or customised exception

Exception :

- An Exception is an unwanted or unexpected condition which disturbs our normal flow of execution.

Ex:

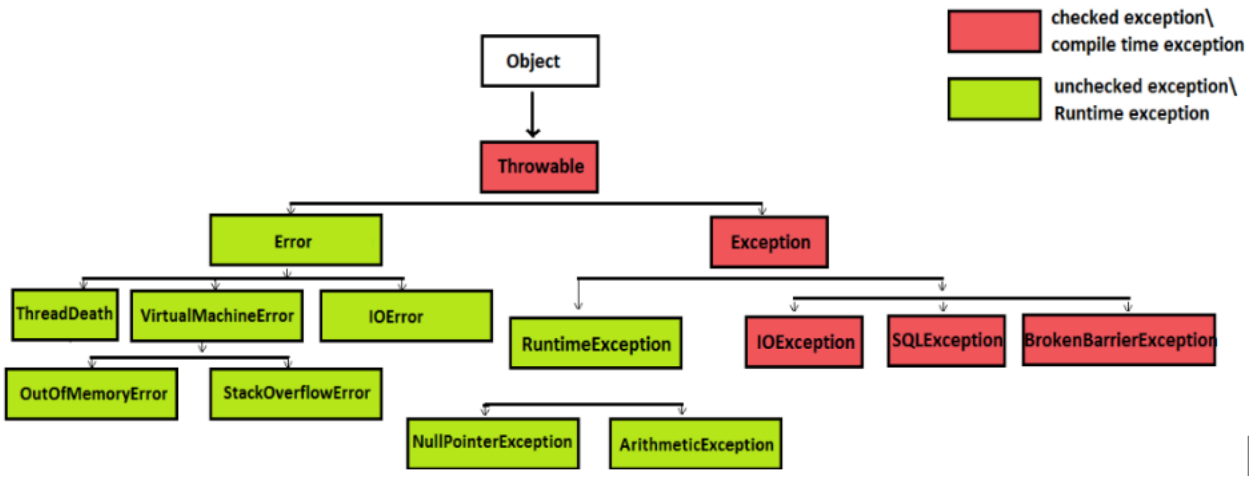
1.CoronaException
2.lockDownException

- Once an Exception occurs, the remaining part of the program will not be executed.
- So, it is our responsibility to handle the exception.
- Exception handling doesn't mean we are resolving an exception, it is just like providing an alternate solution so that even though an exception happens our program should work properly.

Exception Hierarchy:

- Object class is a superclass to all the predefined and user defined classes of java.
- Throwable class is a super class to "Exception" class and "Error" class.
- Exception class is a super class to RuntimeException class and other Exception classes.
- All the Exception classes belong to the java.lang package.
- Depending on Hierarchy, Exceptions are divided into 2 types -

1.Checked Exception (Compile time Exceptions)
2.Unchecked Exception (Run time Exceptions)



Checked Exception

• Exceptions which are checked(identified or found out) during compile time by the compiler, such types of exception are called Checked Exceptions. (or) Exception classes which are directly inheriting Exception Class except RuntimeExceptionclass is called as checked exception.
• Checked Exceptions are also called Compile time Exceptions.
• Examples(Classes) of Checked Exceptions are :-
-InterruptedException
-ClassNotFoundException
-SQLException
-FileNotFoundException

Unchecked Exception

• Exceptions which are checked(identified or found out) during Runtime or execution time, such type of exception are called as Unchecked Exceptions.
• Incase of Unchecked Exception our program will at least compile successfully.
• Unchecked Exceptions are also called Runtime Exceptions.
• RuntimeExceptionclass is a superclass to all UncheckedException classes.
• Examples(Classes) of Unchecked Exceptions are :-
-ArithmeticException
-ArrayIndexOutOfBoundsException
-NullPointerException
-StringIndexOutofBoundsException
-ClassCastException
-NumberFormatException

Error :

• An Error is an irrecoverable Condition i.e, if error occurs it is not under programmers control to get over it.
• For Ex: if we develop any program whose size is 4gb but our system's storage is 3gb so such condition is not in programmers control and such situation is referred as Error.
• Examples(Classes) of Error are :-
-StackoverFlowError
-VirtualMemoryError
-404pagenotfound

***Differences between Error and Exception

Error	Exception
1. An error is caused due to lack of system resources.	1. An exception is caused because of some problem in code.
2. An error is irrecoverable i.e, an error is a critical condition cannot be handled by code of program.	2. An Exception is recoverable i.e, we can have some alternate code to handle exception.
3. There is no ways to handle error.	3. We can handle exception by means of try and catch block.
4. As error is detected program is terminated abnormally.	4. As Exception is occurred it can be thrown and caught by catch block.
5. There is no classification for Error.	5. Exceptions are classified as checked and unchecked.
6. Errors are define in java.lang.Error package	6. Exceptions are define in java.lang.Exception package

What happens when an Exception occured ?

```
class Sample
{
    public static void div()
    {
        int a=10,b=0,c; c=a/b;//10/0
        System.out.println("Exception Occurred");
        System.out.println(c);
    }
    public static void main(String args[])
    {
        div();
    }
}
```

- In the above program execution begins from the main method and it calls div().
- In div() there is an unexpected statement i.e c=10/0 when this statement is encountered div() creates an Exception object which includes -

Name:

description:
location:
and handover it to JVM. Now JVM will check if there is any exception handling code present in div() since there is no exception handling code present in div() it checks with the caller method in the above program i.e, main(). So it checks in main() if there is any exception handling code present or not. Since, there is no exception handling code present in main() also, JVM will call the default Exception handler i.e printStackTrace()(method) and that default exception handler provides the complete information of Exception.

i.e, Exception in thread "main" java.lang.ArithmeticException: / by zero at Sample.div(Sample.java:6) at Sample.main(Sample.java:12)

• Once an exception occurs, the remaining part of a program will not be executed and ending up with abnormal termination.
• So, if we want the above program to be executed and have normal termination then, we have to handle the Exception.
• Exceptions can be handled by using try and catch blocks.

Ex 1: Exception didn't occurred, so except catch block all normal statements will be executed

```
class Sample
{
	public static void div()
	{
		int a=10,b=10,c;
		try {
			System.out.println("Starting of try");
			c=a/b;//10/10
			System.out.println("This is a try block");
		} catch(ArithmeticException e) {
			System.out.println("Exception is occurred and handled");
		}
	}
	public static void main(String args[])
	{
		div();
		System.out.println("End of main");
	}
}
```

Ex 2: Exception occurred in try block and JVM have executed

corresponding catch block after exception statement

```
class Sample
{
	public static void div()
	{
		int a=10,b=0,c; try
		{
			c=a/b;//10/0
		}
		catch(ArithmeticException e)
		{
			System.out.println("Exception is occurred and handled");
		}
	}
	public static void main(String args[])
	{
		div();
	}
}
```

Ex 3: Exception occurred in try block and JVM have executed corresponding catch block after exception statement no other statements executed.

```
class Sample
{
	public static void div()
	{
		int a=10,b=0,c;
		try
		{
			System.out.println("Try starts");//executed c=a/b;//10/0
			System.out.println("Try block");//unexecuted
		}
		catch(ArithmeticException e)
		{
			System.out.println("Exception is occurred and
			handled");//executed
		}
	}
	public static void main(String args[])
	{
```

```
            div();
            System.out.println("main ends");//executed
        }
}
```

Q. Can we write any statement between try and catch block ?
No, we cannot write any statement between try and catch block. Immediately after a try block there should be catch or finally block.
Q. Can we write only try block without catch block ?
No, a try block should always be followed by either catch or finally block.
Q. If we don't know the exception type, what type should we mention in the catch block ?
When we do not know Exception type, we can mention it as ExceptionClass type or Throwable type. Because, Exception is a super class to all the class and during upcasting we studied that superclass can hold reference to subclass objects
Single try with multiple catch blocks
Yes, it is allowed to write a single try with multiple catch blocks but it should be most specific to most general.

```
class ExceptionEx {
    public static void main(String args[]) {
        int a = 10, b = 0, c;
        int d[] = new int[3];
        try {
            d[3] = 55;// beyond declared size
        }
        // most specific catch block
        catch (ArrayIndexOutOfBoundsException f) // AIOBE f=new AIOBE()

        {
            System.out.println("Exception is caught for array");
        }
        // general catch block
        catch (Exception e) // Exception e=new AIOBE()
        {
            System.out.println("Exception is caught");
        }
    }
}
```

output:

```
Exception is caught for array
```

In the above program there is ArrayIndexOutOfBoundsException occurred and if

we have written multiple catch blocks we always have to write specific catch block i.e, catch block which contains AIOUBException reference first then we can write general catch block means catch block which have Exception class reference.
• For the above program, we cannot define catch blocks as most general to most specific. If we do, we will get a compile time error.

finally block

• finally is a block which will get executed irrespective of
1.exception occurred or not
2.exception occurred and handled
3.exception occurred and not handled

1.exception occurred or not :

```
class ExceptionEx {
    public static void main(String args[]) {
        int d[] = new int[3];
        try {
            d[2] = 55;
        } catch (ArrayIndexOutOfBoundsException f)// AIOBE f=new AIOBE()
        {
            System.out.println("Exception caught for array");
        } finally {
            System.out.println("Finally block");
        }
    }

}
```

output

```
Finally block
```

2.exception occurred and handled

```java
class ExceptionEx {
    public static void main(String args[]) {
        int d[] = new int[3];
        try {
            d[3] = 55;// beyond declared size
        } catch (ArrayIndexOutOfBoundsException f)// AIOBE f=new AIOBE()
        {
            System.out.println("Exception caught for array");
        } finally {
            System.out.println("Finally block");
        }
    }
}
```

output

```
Exception caught for array
Finally block
```

3.exception occurred and not handled

```java
class ExceptionEx {
    public static void main(String args[]) {
        int a = 10, b = 0, c;
        int d[] = new int[3];
        try {
            d[3] = 55;// beyond declared size
        } finally {
            System.out.println("Finally block");
        }
    }
}
```

output

```
Finally block
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
        at com.example.demo.ExceptionEx.main(ExceptionEx.java:8)
```

Basically, "finally" is used to keep an important code which should not be skipped at any condition like closing of database connection or closing of opened file etc.

Valid Combinations

```
1. try{}
 catch{}
 finally{}

2. try{}
 finally{}

3. try{}
 catch{}
 catch{}
 finally{}

4. try{}
 catch{}
 finally{}
 try{}
 catch{}
 finally{}
```

Invalid Combinations

```
1.   try{}
     catch{}
     finally{}
     finally{}

2.   catch{}
     finally{}

3.   finally{}
     try{}
     catch{}
```

Throws Keyword

```
class ExceptionEx {
    public static void main(String args[]){
        System.out.println("Go to sleep");
        Thread.sleep(1000);
        System.out.println("Awake");
    }
}
```

In the above program we have called sleep() of thread class means, we are making the current thread i.e, the main thread to go to sleep. Which means main() should

stop execution. Once sleep() is invoked for given amount of time i.e, 1000 milliSeconds.
• But, When it is in a sleeping state, there will be a chance of other threads trying to interrupt the main thread from sleeping. So, that is why for above program we will get the Exception as

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Unhandled exception type InterruptedException

        at com.example.demo.ExceptionEx.main(ExceptionEx.java:6)
```

Conclusion :

In case of checked Exception, the user has 2 options -
1. Caught the Exception : use try and catch block
2. Declare the Exception : use throws keyword
• throws keyword is used to declare or report a checked Exception.
• throws keyword is used with method declaration. public void fly() throws ExceptionName
• When we use throws keyword, we are indicating that the current method will not handle exception rather than calling method or its caller will handle the exception.

```
class ExceptionEx {
    public static void main(String args[]) throws InterruptedException{
        System.out.println("Go to sleep");
        Thread.sleep(1000);
        System.out.println("Awake");
    }
}
```

In the above program, main method declared an Exception using throws keyword that means it is telling that i won't handle exception rather it is responsible of my caller to handle it.
• So, in the above program the caller of main() is JVM. And, this process where the current method is not handling exceptions and telling the caller to handle. It is called Exception Propagation.

```
class ExceptionEx {
    public static void main(String args[]) {
        throw new ArithmeticException("My Exception");
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: My Exception
        at com.example.demo.ExceptionEx.main(ExceptionEx.java:5)
```

```java
class ExceptionEx {
    public static void check(int age) {
        if (age < 18) {
            throw new ArithmeticException("He is small kid - enjoying life");
        } else {
            System.out.println("cast a Vote");
        }
    }
    public static void main(String args[]) {
        check(15);
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: He is small kid - enjoying life
        at com.example.demo.ExceptionEx.check(ExceptionEx.java:6)
        at com.example.demo.ExceptionEx.main(ExceptionEx.java:12)
```

User defined Exception (or) Customised Exception

When a predefined exception does not fulfil our requirement, we will go for user-defined exceptions i.e we can create our exceptions. Such type of exception is called as User define Exceptions (or) Customised Exceptions.

Rules for creating User defined Exceptions:-

- create our Exception class and that class should be or must be extending either throwable (or) Exception (or) RuntimeException classes Preferable to extend RuntimeException
- Define constructor whenever required.
- Throw Exception as per our own requirement.

```java
class NotEligibleException extends RuntimeException // Ex-1
{
    public NotEligibleException(String msg) {
        System.out.println(msg);
    }
}
public class ExceptionEx {
    public static void main(String args[]) {
        float percentage = 56.5f;
        if (percentage < 60) {
            throw new NotEligibleException("Not Eligible for drive");
        } else {
            System.out.println("Register before end of the day");
        }
    }
}
```

```
Not Eligible for drive
Exception in thread "main" com.example.demo.NotEligibleException
        at com.example.demo.ExceptionEx.main(ExceptionEx.java:14)
```

Explanation:
• Execution started from main method, under that i gave condition, if that condition satisfied, i am creating an explicit Exception object with details as-
Name: NotEligibleException
Description: NotEligible for drive
Location: User.main
• Once Exception objects created it calls to NotEligibleException constructor and pass msg as argument
• under that constructor we print msg, so it prints that information.

```java
class AgeGapException extends RuntimeException // Ex-2
{
    public AgeGapException(String msg) {
        System.out.println(msg);
    }
}

public class ExceptionEx {
    public static void main(String args[]) {
        int bage = 10, gage = 27;
        if (bage < 15 && gage >= 27) {
            throw new AgeGapException("He is small kid enjoying life");
        } else {
            System.out.println("Go out with girl friend and destroy life");
        }
    }
}
```

```
He is small kid enjoying life
Exception in thread "main" com.example.demo.AgeGapException
        at com.example.demo.ExceptionEx.main(ExceptionEx.java:14)
```

Explanation:
• Execution started from main method, under that i gave condition, if that condition satisfied, i am creating an explicit Exception object with details as-
Name: AgeGapException
Description: He is small kid enjoying life
Location: Demo.main
• Once Exception objects created it calls to AgeGapException constructor and pass msg as argument
• under that constructor we print msg, so it prints that information.

****Differences between throw and throws**

throw	throws
throw keyword is used to create Exception object explicitly	throws is used to declare the Exception
throw keyword is used inside the method	throws keyword is used with method declaration
Syntax: throw new ExceptionName(Excp description); Ex: throw new ArithmeticException("MyExcept");	Ex: method declaration Exception Name public void fly() throws InterruptedException
throw keyword is mainly used for Userdefine exception	throws keyword is mainly used for checked exception
Using throw keyword, we can throw only one exception at a time	Using throws keyword, we can declare multiple exceptions at a time
throw new MinBalException("Zero");	public void check() throws InterruptedException,SQLException

Q. WAP to demonstrateuser defne exception create InsufficientBalanceException class create cust class if(withdrawlamt > avabal)---->throw InsufficentBalException else---------->collect the amt

```java
import java.util.Scanner;

class InsufficientBalException extends RuntimeException {
    public InsufficientBalException(String msg) {
        System.out.println(msg);
    }
}

public class Customer {
    public static void main(String args[]) {
        System.out.println("Enter available balance in account : ");
        Scanner s = new Scanner(System.in);
        double avlbal = s.nextDouble();
        System.out.println("Available balance : " + avlbal);
        System.out.println("Enter the amount to withdraw : ");
        double wdramt = s.nextDouble();
        if (wdramt > avlbal) {
            throw new InsufficientBalException("Exceeding the limit in the account");
        } else {
            System.out.println("Collect the amount");
        }
    }
}
```