OOPS Concepts:-

Encapsulation :- Wrapping data members & its functions in to a single unit. We achieve in java using Class. In class we have methods and variables, these are referred by object of a class.

If Creating a Class with Private variables & these Private Variables can be Accessed using Public setters & getters methods

Setter methods are used to set values for private variables(Class Level) of a class.

Setter method Return Type will be always void.

Getter Method return type will be always the return type of its variable

Q)What are OOPS Concepts? How did u achieved in u r project?

A) in java we have 4 types of oops concepts

Q) What is encapsulation? How did u achieved in u r project?

Encapsulation :- Wrapping data members & its functions in to a single unit. We achieve in java using Class. In class we have methods and variables, these are referred by object of a class.

1. Abstraction :-

Hiding the Implementation of the functionality. It Tells how it is works & but it doesn’t tell how it was Implemented.

Ex;- clicking on login button(we don’t know how it is validating the Authentication)

Ex:- clicking on Submit button(we don’t know how all the data is saved & submitted in the backend)

This is Achieved by Interfaces & Abstract Classes.

100% Abstraction can be Achieved with Interface

100% Abstraction cannot be achieved with Abstract Class because it might/can contain defined methods because of which 100% Abstraction is not possible

|  |  |
| --- | --- |
| Abstract Class | Interface |
| It Contains Abstract Methods & Defined Methods & Constants | It contains only Abstract Methods & Constants(public static final) |
| Abstract Keyword has to applied at class level & Method Level for Abstract methods | Abstract Keyword is optional at Interface level & method level because JVM will anyways provide . |
| Abstract class can contain Constructor but it will be used by its subclasses | Interface cannot contain Constructor because it will give compilation error if we add Constructor |
| Class &Abstract Class will not participate in Multiple Inheritance | Interface can be used in multiple Inheritance. |
|  |  |
|  |  |
|  |  |

Similarity

object Cannot be created for Both Abstract class & Interface.

Abstract class objects can be create by using its sub classes.

Interface methods are accessed by using its sub class objects.

Q) What is abstraction? How did u achieved in u r project?

Hiding the Implementation of the functionality. It Tells how it is works & but it doesn’t tell how it was Implemented. We can use this feature an interfaces and abstract classes with abstract methods . by looking at this abstract methods we don’t know how it was implemented, because these methods are overridden in it’s sub classes.

Q) What is difference b/w interfaces and abstract classes?

Refer the above table.

Q) how can u solve diamond problem in inheritance?

Using interfaces. see the hybrid inheritance in diagram .

Inheritance :- Inheritang the Properties and behaviours(variables and methods) from Parent to its children

1. InHeritance :- Inheriting the variables& methods from SuperClass to its SubClasses(Parent to Child)

Private methods & private variables cannot be accessed/Visible in SubClasses.

Inheritance can be achieved through **extends** keyword between classes(A,B)& Interfaces(I1,I2)

Ex:- **Class A extends B**

Interface I1 extends I2

Class A can Implement Interface I1

Ex:- Class A Implements I1

Interface I1 cannot Implement or Extend Class A

Interface I1 implements class A (Not Possible)

**Single Inheritance:-**

We can Create objects of SuperClass with SuperClass Reference so that we can access all the Public methods in SuperClass.

We can Create objects of SubClass with SubClass Reference so that we can access all the Public methods in SubClass & SuperClass

If we Create objects of SubClass with SuperClass Reference then we can access all the Public methods of SuperClass But we will not be able to access methods from Sub Class.

**Multiple inheritance**  in the below diagram A and B are interfaces C is sub class of A and B.

**Note**: If a class (C) implements an interface(I) it has to override all the methods of interface.

if we don’t override any method of the interface It will gives compilation error.

If an abstract class (C) implements an interface(I) overriding the methods of interface is optional.

**Hybrid inheritance** is the combination of more than one inheritances

In Hybrid Inheritance, A,B,C should be Interfaces because if A is Class B & C cannot implement class A.

If a Class A extends ClassB and implements Interfaces I1,I2 we should write the syntax as below:-

Class A extends B implements I1,I2

Class A implements I1 extends B ------not possible throws compilation error

Q) What is inheritance? How did u achieved in u r project?

Inheriting the variables& methods from SuperClass to its SubClasses(Parent to Child)

In our project generally we extends some abstract classes or interfaces



4.Polymorphism :-

Same Action Can be performed in multiple ways

We can achieve this feature in java using methods

1. **Compile time Polymorphism** :-polymorphism which occurs at compile time is called Compile time Polymorphism .This can be Achieved by Method Over Loading Concept.

Method Over Loading Concept :- It is performed in a Class

Methods Name are Same but Signature should be different.

Method Signature:- Number of arguments, data types of Arguments, order of the Arguments

If two methods have same name with same signature, Access Specifiers & return types doesn’t make a difference.

1. **Run time Polymorphism**:- polymorphism which occurs at run time is called run time Polymorphism . This can be Achieved by Method Over Riding Concept.

**Method OverRiding** means redefining body of a method .It can be Achieved using Inheritance concept.

Methods Name are Same & Signature are also Same . Along with these method return types & Access Specifies should be also the same.

Q) What is Polymorphism ? How did u achieve in u r project?

A) Same Action Can be performed in multiple ways. we have 2 types polymorphism .

In our project we have overloaded methods and overridden methods

**Annotation:** Annotation is a special type of interface, we can create our own Annotations also.

Examples : **public** **@interface** Override { }

**public** **@interface** Deprecated { }

**Constructor**:-

Constructor is used for Allocating Memory for the variables and methods of Class.

Rules for Constructor:-

1. Constructor name should be exactly same as ClassName(CaseSensitive)
2. Constructor will not have return type(like void, int, String)

Default Constructor( 0 Arguments Constructor) is optional which is provided by JVM

Class can Have Default Constructor & OverLoaded Constructors.

For OverLoaded Constructor, Signature should be different.(Number of arguments, data types of Arguments, order of the Arguments)

In a class if we override overloaded constructor(1 or 2 arg constructor) and if we try to call default constructor(creating object with default constructor) it will give compilation error because jvm will not provide default constructor in this case

We can add return statement inside a constructor it does not give compile time or runtime error

Class A

A(){

………………….

return;

}

Q) What is the use of constructor ?

Constructor is used for Allocating Memory for the variables and methods of Class.

Q) in how many ways we can assign values to private variable in class?

A) 2 ways {using setter methods, using constructor }

It depends on the requirement to choose constructor approach or setter methods approach.

Note: check the execution flow of constructors in inheritance.

***Static* :-** in java, Static means one time memory allocation per jvm. JVM Allocates memory for static variables Static methods & static blocks at compile time. Static is a Keyword which is used at Variable level & method level but not at the Class Level. Which means class can contain static variables ,Static methods & static blocks also.

JVM Allocates memory for static variables Static methods & static blocks at compile time so that we can access these without creating objects(using Class name, we can access these.)

If we try to access these with objects name it does not give any compile time or run time error. It just gives Warning.

Static methods cannot be Overridden in its subclasses because the methods are at class level but they can be accessed in Subclasses.

A Class can contain instance variables, static variables ,constants, instance methods, static methods, constructors, static blocks & instance blocks.

If a Class contains all the above details then it will be executed in the following order:-

1. All the Static Blocks in the order
2. When we create a object of the class all the instance blocks in the order are executed first
3. If constructor is overridden in class that will be executed.
4. Methods will be executed if they are called.

Abstract class can contain static method , abstract method ,define method (instance method).

If we give static for an abstract method it will throw compilation error. Because abstract method memory is allocated at runtime but static method memory loads at compile time.

Q) what is the use of static in java?

A) in java, Static means one time memory allocation per jvm. JVM Allocates memory for static variables Static methods & static blocks at compile time. We cannot add static at class level. if we add, It will gives compilation time error.

Q) can we override static methods? What happens if we add same method of super class in its sub class as static and non static, why?

A) no. if we add same method of super class in its sub class as static, it will not give any error. it will be treated as separate method of **sub class.**

if we add same method of super class in its sub class as non static, it gives compilation error because, static methods are class level methods, which means methods are loaded at compilation time. But in method overriding memory will be allocated for methods at runtime.

**Final** is a Keyword, it is used to create constants and it is used at Variable level & method level & Class Level . In a Class if a variable is declared as Final the value will not be changed once it is assigned.

We can not assign final variable multiple times as below

Class A {

Final String s = null;//the value is assigned as null,can not be changed any where.

Final int I;//this is compilation error

Final Methods cannot be overridden in its subclasses but they can be accessed in Subclasses.

Final Classes cannot participate in inheritance that means we cannot extend by final class.

Q) what is final, why final methods cannot be overridden?

A) final is a Keyword, it is used to create constants and it is used at Variable level & method level & Class Level . In a Class if a variable is declared as Final the value will not be changed once it is assigned.

If we override final methods, it will give compilation error. Because Method overriding means redefining body of a method . but if we declare a method as final, we cannot redefine the body.

Q) what are the rules for final class ?

A) we cannot make final class as super class to any other class. It means final class doesnot have sub classes. But final class can extend any other class(it can be normal class, abstract class, or interface).

**this :** this isa keyword which is used to refer current class object . we can access current class methods and variables using **this**. This is used at variable level and method level and **this()** will be treated as default constructor.

Generally **this()** is used as first statement in a constructor otherwise it will give compile time error.

**super** super is a keyword which is used to refer super class object. we can access super class methods and variables using **super.**

Generally **super()** is used as first statement in a constructor otherwise it will give compile time error

**Note** :in constructor we can add either this() or super() but not both.

**Access specifiers:** we have4 types of access specifiers in java which is used at variable level method level ,constructor level and class level

**Private**: access only inside the class

**Defau**lt: if we don’t give any access specifier it will be treated as default. This is package level specifier that means we can access any variable or method of a class p1.A(p1 is package ) in any class inside the package p1

**Protected**: we can access a variable of a class p1.A(p1 is package ) in any of the class inside the package p1 , any of its sub packages class(p1.p2.B)and any other package classes(p3.C) with extends keyword. that means class(B or C) has to extend the protected Variable Class(A).

If we access protected variables in outside of the class with object name it will give compilation error.

**Public:** we can access public variables or methods in any other class of any package

**Wrapper Classes:** in java we have primitive data types and wrapper Classes also .java supports user defined variables .

We can convert primitive data types to wrapper classes and vice versa

Using wrapper classes we can achieve 100% object orientation in java.

Examples: int ,char , boolean ,float etc.. are primitives

Integer,Character ,Boolean,Float etc….are WrapperClasses

Class A{}

A a1;// here a1 is user defined variable

By default primitive data types contain some default value

Int (0) , boolean (false) ,float (0.0f)

By default all wrapper class, String and user defined class values are “null”.

**Note:** in java Class if we want to exit at any time we use **System.exit(0).** When code execute this statement control will come out of jvm.

Q) what is Boxing , AutoBoxing And unboxing

A) Boxing , AutoBoxing both are same

**Autoboxing and Unboxing:**

The automatic conversion of primitive data types(eg int) into its equivalent Wrapper type (eg Integer) is known as boxing and opposite operation is Unboxing.

**Conditional Clauses:**

1. If
2. If-else
3. Nested- if-else(if-elseif-elseif…..else)
4. Switch

If: if is a conditional statement it executes when the condition is true.

In if condition it is optional to put “{ }” if we have only one statement. Otherwise we have to put” { }” to execute the block of statements.

If a method has only if conditions all the if blocks will be checked even if condition is satisfied at the beginning. but in nested if else, blocks will be checked until the condition is true. It means once he condition is true remaining conditions in else if or else block will not be executed.

**Break:** generallybreak is used inside the loops or conditional statements. Once break is executed, control will come out of the loop.

For(){

For(){

If(condition){

Break;

}

}

Block of statements

}

In the above syntax, once if block is executed, control will come out of inner for loop, but outer for loop will be executed as it is.

**Return :** generally return is used in side methods. it is used for returning a value from the method. it means once return statement is executed in side a method at any place, control will come out of the method.

Void m1(){

for(){

If(condition){

Return;

}

}

//some stmt to be executed

}

In the above syntax when if condition satisfied, return statement will be executed. Control will come out of the method.

Q) what is the use of break and return statements?

A) generallybreak is used inside the loops or conditional statements. Once break is executed, control will come out of the loop.

Return is used for returning a value from the method. it means once return statement is executed in side a method at any place, control will come out of the method.

**Loops :** if we want to execute some statements repeatedly we use loops.

1. **while** : in while loop statement will be executed inside the loop until the condition is false

syntax:

< initialization of variable>

While(<condition>){

Statements to be executed

<increment / decrement of variable >

Statements to be executed

}

1. **do while:** in do while the loop will be executed once without checking condition. The condition will be checked at the end of first execution. it means loop will be executed from second iteration only if the condition in while() is true.

Syntax

< initialization of variable>

Do{

Statements to be executed

<increment / decrement of variable >

Statements to be executed

}while(<condition>)**;**

Q) Deference b/w while and do while

A) In while loop initially it checks the condition. if condition is true then only while loop is executed.

In do while loop initially loop will be executed once without checking condition.

**note** : **” ;”** ismandatory in do while loop.

1. For loop:

in formal for loop initialization , condition ,increment /decrement will be present in one line.

Syntax:

For(initialization ; condition ;increment /decrement){

Statements to be executed

}

Or

<initialization>

For(;condition ;){

Statements to be executed

<increment / decrement of variable >

Statements to be executed

}

For loop execution

In for loop,

Step1 : variable will be initialized,

Step 2 :condition will be checked.

Step 3: If condition is true, loop will be executed.

Step 4 : After loop is executed, increment/decrement operation performed.

Step 5: in step4 operation, variable holds the increment/decrement value. With this value step2,step3,step4 will be performed until step 2 fails(condition is false)

**For-each loop:** generally for-each loop is used to execute array or collection of values.

Syntax:

Int a[]={1,2,3,4};

For(int I:a){

Syso(i) ;

}

In the above syntax, “i” refers each element in the array(a).

**Java.util.Scanner:** it is a class which is used to read all types of values from console at runtime.

**Exception Handling:** in java, Exception is a class which is super class for compile time and runtime errors. And Exception is sub class for Throwable .

in javawe have 2 types of exceptions

1. Java.lang.Error : we cannot handle this exceptions in java.
2. Java.lang.Exception: we can handle this exceptions in java.

For all the exceptions java.lang.Throwable is the super class

Java.lang.Exception is categorized into 2 types

1. Compiletime errors(java.lang.Exception) Or checked exceptions .
2. Runtime errors(java.lang.RuntimeException) Or unchecked exceptions.

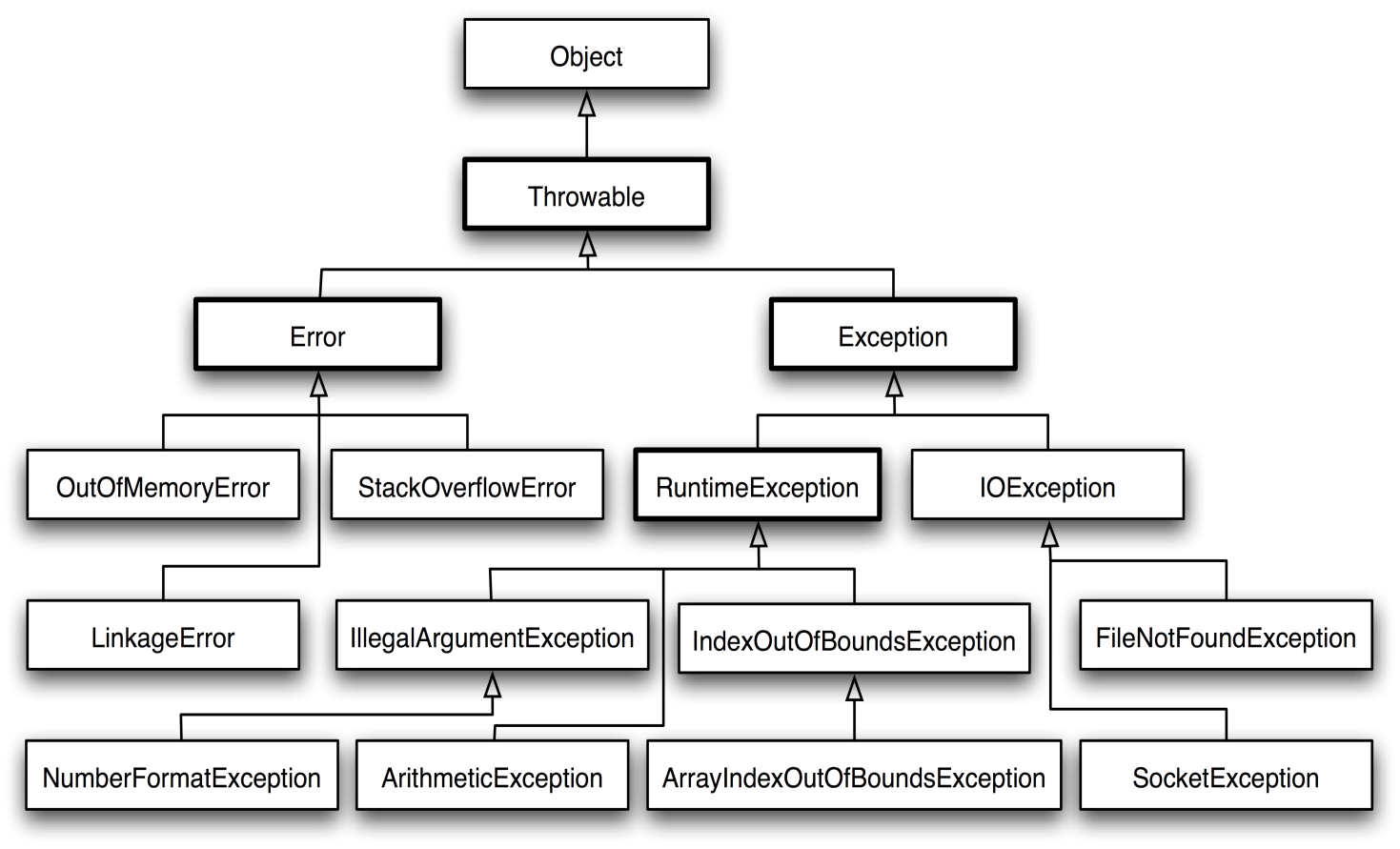
Here RuntimeException is the subclass of java.lang.Exception

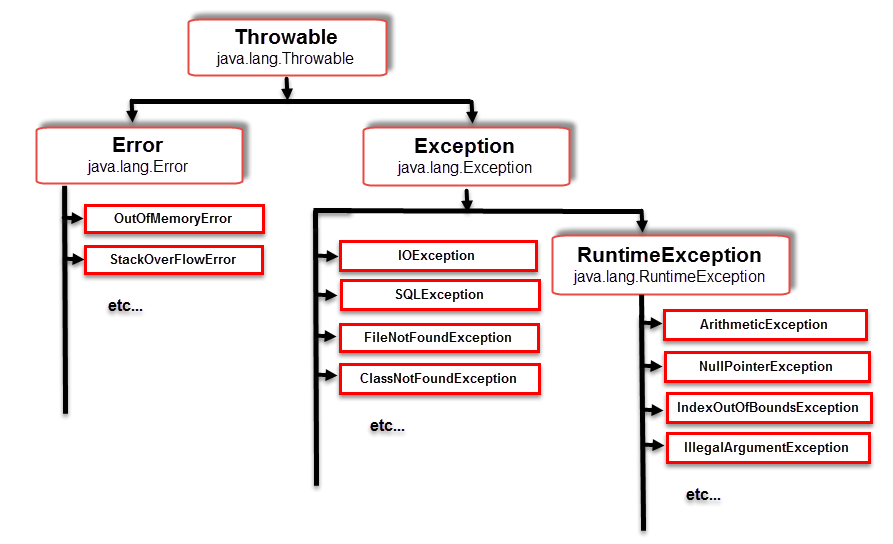
**Compile time errors**: Exceptions which occur at compilation time. Including java.lang.Exception and it sub classes (not java.lang.RuntimeException, and its sub classes) are come under compilation errors. In java we have to handle these exceptions at compilation time.

**Runtime errors :** Exceptions which occur at execution time(runtime). java.lang.RuntimeException, and its sub classes are come under runtime errors.

If we donot handle runtime errors there is no problem at compile time.







We handle exceptions using try , catch and finally blocks. We use throw and throws in handling the exceptions.

In try block generally we keep **problematic stmts** (stmts which can give compile time or runtime errors). After try block we have to write catch block. If we don’t write catch block it will give compilation error.

if we keep any other stmt b/w try and catch blocks, it will give compilation error.

In catch block , we try to write some userdefined stmts, when exception occur. These stmts are used for logging purpose.

Writing finally block is optional. Generally we use finally block, if we are dealing with file read/write operations or data base operations.

Q) explain the execution flow of try, catch and finally blocks when we return a value from try block and when we write System.exit(); in try block?

* If we write **return** stmt in **try block** and the try block has catch and finally blocks, it executes try block first , when control reaches to return stmt it execute finally block. After that control comes to return stmt in try block and exit the method. If we write **System.exit(2);** in try block, when control reaches this stmt, control will come out of jvm. it will not execute finally block.

In java ,after try block we can write catch or finally blocks. But before catch block, only try block is allowed.

**Throw and Throws:** throw and throws are keywords. Generally throw is used inside a method or constructor.

**Syntax:** throw new <exception-class-name>();

**Ex1:** public void m1(**String s** )throws **Exception**{

If(!s.equals(“string1”)){

//Throw new **Exception**();

Throw new **Exception**(“input is not matched”);

} }

**Ex2:** public void m2(**String s** ){

If(!s.equals(“string1”)){

Throw new **RunTimeException**();

} }

If we compare ex1and ex2, ex1 method m1() (code inside method m1() ) is throwing compilation error, we have to handle at compilation time(from where m1() is called).

Ex2 method m2() (code inside method m2() ) is throwing runtime error, handling these exception is optional.

Generally we use **throw** for user defined exceptions.

In java we use both predefined exceptions(exceptions which are provided by jvm). And user defined exceptions(which are provided by user).

If a piece of code is throwing runtime exception it will not be visible at compile time. Handling runtime exceptions(using try –catch blocks) is not a good practice.

A try block can be followed by one or more catch blocks. So once Exception occur in try block , the catch blocks will be executed sequentially. The catch blocks order is from child exception class to parent exception class. if the order is changed it throws compile time error.

Q) difference b/w throw and throws?

|  |  |
| --- | --- |
| **Throw** | **throws** |
| Throw is used inside a method. | Throws is used at method heading. |
| Syntax: throw new Exception(“Exception”); | Syntax: public void m1() throws Exception, IOException{ //method body  } |
| We can throw exception in constructor. | We can write exception at constructor header level using throws. |
|  |  |
|  |  |

If a method throw any compile time error inside a method, we will write the exception name at method heading level beside throws. Example is

Class A{

Public void m1() throws Exception{

Try{

//method stmts

}catch(IOException e) {

Throw new Exception();

} } }

If we write any stmt after throwing exception(using **throw** key word) inside a method we can not write any other stmt including return. It gives compilation error (unreachable error).

If a method or constructor is throwing exception, where ever we are calling the method or constructor all the places we have to handle by throwing the exception again or using try –catch block.

Static or instance blocks can not throw exception. If static or instance block code is throwing compilation error, we have to handle the exception using try-catch.

**User defined exception:** user defined exception is a normal class, but it should have the below condition.

The class should extend checked(Exception) or unchecked(RuntimeException) exception class.

Class C1 extends Exception{

}

Q) What are the Differences b/w final, finalize,finally?

A) **final** is a keyword and it is used to create constants. **Finally** is a block it is used in Exception scenarios. It means the stmts in finally block will be executed even if exception occur in try or catch blocks. **Finalize** is amethod in Object class and it is used in garbage collection process.

Q) what is marker interface, Why we use it?

A) Marker Interface in java is an empty interface. It means interface does not have constants and methods. It tells the JVM that the class implementing marker interface will have some special behavior. Examples **Cloneable , Serializable.**

**Serializable:** if a class is implementing serializable interface, The serializable interface provides a mechanism of object serialization where an object is represented as a sequence of bytes. Object serialization writes the object intostream.  
  
it means the class can be transfer through FTP(eg: we can write the class into a file and we can read the object from the file )**.**

If we do not implement serializable interface and if we try to write the object to a file, it will throw runtime error(java.io.NotSerializableException**).**

**Cloneable:** if a class implements Cloneable interface, we can create the clone of an object(it means it will create a copy of object ). In order to get the clone of an object, it is required to type casted into its appropriate type. If a class is not implementing the cloneable interface, and we try to clone that object we get a CloneNotSupportedException .

We have to override clone() in our class. When we are calling java.lang.Object.clone() (super.clone ) in the overridden clone(), it may throw checked exception (CloneNotSupportedException).

We have to typecast the clone() to our own class.

Factorial: product of a number from 1 to it self.

5!=5\*4\*3\*2\*1

Palindrome: reverse of a number or word(String) equals to the same.

1441,12121,madam,amma.

Armstrong: a number should be equals to the sum of the cube values of the digits.

153= (1\*1\*1)+(5\*5\*5)+(3\*3\*3)

Prime number: prime number is divisible by 1 and it self (not by any other number). Prime numbers starts from 2.

2,3,5,7,11,13,17,19,23,etc..

Fibonacci series: it starts from 0,1. Next digit will be equals to the sum of the 2 left side numbers.

Eg: 0,1,1,2,3,5,8,13,21,34,55 etc…

**Factory method :** it is used to provide objects. In java we have static and instance factory methods.

**Q) What is singleton and how can we achieve?**

**A) Singleton class:** Singleton is a design pattern it is used to create only one object per jvm. We can achieve singleton with a static factory method and a private constructor.

**Q) what is String immutability in java?**

A) String immutability means once the value is assign to a String, the value of the memory can not be changed. it means once we assigned a value to String and we can not append another String for the same memory. If we append another String, it will be stored in new memory,but it will not override the existing value.

**String,StringBuffer,StringBuilder :** in java String is immutable. Which means once the value is assign to a String can not append another String for the same memory. If we append another String, it will be stored in new memory.

To over come String immutability StringBuffer,StringBuilder are used.

**Q) Difference b/w StringBuffer,StringBuilder?**

|  |  |
| --- | --- |
| **StringBuffer** | **StringBuilder** |
| StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| StringBuffer is less efficient than StringBuilder. | StringBuilder is more efficient than StringBuffer. |

Q) difference b/w .equals()and == operator?

**.equals():**  it verifies Strings value only. It is used to verify 2 objects.

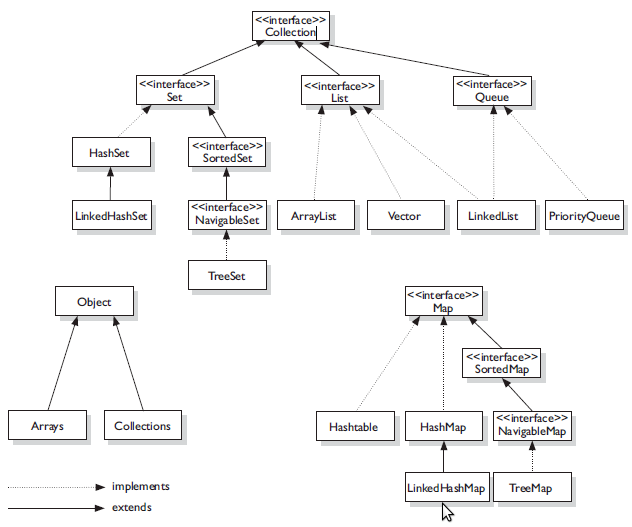
**“==”:** it verifies Strings value and memory.

**Q) What is StringTokenizer?**

**StringTokenizer:** The **java.util.StringTokenizer** class allows you to break a string into tokens(small Strings). It is simple way to break string. To implement this we will create 2 arg constru1ctor for **StringTokenizer class.**

StringTokenizer stringTokenizer = new StringTokenizer(String s1, String delim)

**Collection Freamework :**



In collections we can store 1-dimention elements and 2-dimentional elements . By looking at the above diagram **Collection** and its hierarchy is used for 1-dimentional elements(starts from index 0) and **Map** and its hierarchy is used for 2-dimentional elements(kay-value pairs).

Array is used to store homogeneous data but, Collections support both homogeneous and heterogeneous data(we can add any type{int, char, float, Boolean, String, wrapper Classes, userdefined classes} of data). To overcome this We can create generic collections from java1.5.

Generic collections will support only Class type data, not Premitives

Collection<int> // wrong. it gives compilation error.

Collection<Integer> // correct

**Q) Why we use collections in java?**

A) in java arrays has a fixed size. so if we give more size to an array and store less no of values, memory will be wasted. If we give less size and store more values(more than array size) it will throw ArrayIndexOutOfBoundsException. To overcome this problem java interduced collections which increases the size dynamically.

**Q) difference b/w List and Set?**

|  |  |
| --- | --- |
| **List** | **Set** |
| List allows duplicate values | Set allows unique values |
| List preserves insertion order (it inserts elements in the order we insert. New element will be added at the end). List inserts elements in linear order. | Set does not preserve insertion order(it inserts values in random order following by hash mechanism.) Set inserts elements in random order. |
| We can add any number of null values. add() returns Boolean value(true). | We can add only one null value. If we add more than one null values or duplicate values, it skips adding of duplicate value. But it will not override old value with new value . here Set add() returns Boolean value (false). |
| List implementations: [ArrayList](http://beginnersbook.com/2013/12/java-arraylist/), Vector,[LinkedList](http://beginnersbook.com/2013/12/linkedlist-in-java-with-example/) | Set implementations: [HashSet](http://beginnersbook.com/2013/12/hashset-class-in-java-with-example/), [LinkedHashSet](http://beginnersbook.com/2013/12/linkedhashset-class-in-java-with-example/), [TreeSet](http://beginnersbook.com/2013/12/treeset-class-in-java-with-example/) etc. |
| Iterator and ListIterator are used to traverse List. Iterator suppots forward direction,  [ListIterator](http://beginnersbook.com/2014/06/listiterator-in-java-with-examples/) can be used to traverse a List in both the directions(forward and backward) | We can use Iterator only to traverse Set. |

**Q) methods in List And Set?**

|  |  |  |
| --- | --- | --- |
| **method** | **List** | **Set** |
| Int size() | It comes from Collection. It gives size of the List | It comes from Collection. It gives size of the Set |
| boolean isEmpty(); | It comes from Collection. It returns **true** if List is empty | It comes from Collection. It returns **true** if Set is empty |
| boolean contains(Object o); | It comes from Collection. It returns **true** if List contains specific element.(it searches the given element in the List) | It comes from Collection. It returns **true** if Set contains specific element.(it searches the given element in the Set) |
| Iterator<E> iterator(); | It comes from Collection. It prepares the List to iterate in forward direction. | It comes from Collection. It prepares the Set to iterate in forward direction. |
| Object[] toArray(); | It comes from Collection. It converts List to Array. | It comes from Collection. It converts Set to Array. |
| boolean add(E e); | It comes from Collection. It is used to add the element. it returns true if the element is added. | It comes from Collection. It is used to add the element. it returns true if the element is added. |
| boolean remove(Object o); | It comes from Collection. It is used to remove the element from the List. it returns true if the element is present in List and it is removed. | It comes from Collection. It is used to remove the element from the Set. it returns true if the element is present in Set and it is removed. |
| void clear(); | It comes from Collection. It is used to remove all elements from List. | It comes from Collection. It is used to remove all elements from Set. |
| E get(int index); | It is a method of List. It returns elements based on index. | Set does not have this method. Because Set inserts elements in random order. so we don’t know which element is present in which index. |
| E remove(int index); | It is a method of List. It is used to remove element based on index. | Set does not have this method. Because Set inserts elements in random order. so we don’t know which element is present in which index. |
| ListIterator<E> listIterator(); | It is a method of List. It prepare the List to iterate in both forward and backward directions. | Set does not have this method. |

**Q)how to add, delete, retrieve elements in List?**

A) add(E e) is used to **add** elements. Remove(int index) is used to **remove element at index** position. Remove(Object o) is used to **remove element if it is present**. Clear() is used for **removing all the elements** from the list. If we want to retrieve a single element, we use get(int index). We can retrieve List elements using **formal-for-loop, for-each-loop,** using **Iterator** to travel in forward direction, using **ListIterator** to travel in forward and backward directions.

**Q)how to add, delete, retrieve elements in Set?**

A) add(E e) is used to **add** elements. Remove(Object o) is used to **remove element if it is present**. Clear() is used for **removing all the elements** from the Set. We can retrieve Set elements using **for-each-loop,** using **Iterator** to travel in forward direction.

**Q) List and Set are sub interfaces of Collection. but why we do not have get(int index) in Set, why it is present in List?**

A) get(int index ) is not present in Collection interface. It is introduced in List.

**Q) difference b/w ArrayList and Vector ?**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList methods are **not synchronized**. | Few methods(eg: add(), get()) of Vector are **Synchronized**. |
| Default capacity is [10] ArrayList **increments 50%** of current array size if number of element exceeds from its capacity. | Default capacity is [10]. Vector **increments 100%** means doubles the array size if total number of element exceeds than its capacity. |
| ArrayList is **not a legacy** class, it is introduced in JDK 1.2. | Vector is a **legacy** class. |
| ArrayList is **fast** because it is non-synchronized. | Vector is **slow** because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object. |
| ArrayList uses **Iterator** interface to traverse the elements. | Vector uses **Enumeration** interface to traverse the elements. But it can use Iterator also. |

**Q) difference b/w ArrayList and LinkedList, when we use these two classes?**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. ArrayList is **better for storing and accessing** data. | Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory. LinkedList is **better for manipulating** data. |
| ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| ArrayList internally uses **dynamic array** to store the elements. | LinkedList internally uses **doubly linked list** to store the elements. |

**Q) difference b/w HashSet and SortedSet? difference b/w HashSet and TreeSet?**

|  |  |
| --- | --- |
| **HashSet** | **SortedSet (or) TreeSet** |
| HashSet store elements in random order. | SortedSet store elements in sorted order(by default ascending order). |
| HashSet allows one null value. | SortedSet does not allow null value. If we add null value, it throws **NullPointerException.** |
|  |  |

**Q) difference b/w HashSet and LinkedHashSet?**

Both HashSet and LinkedHashSet allow one null value. And both store unique values.

|  |  |
| --- | --- |
| **HashSet** | **LinkedHashSet** |
| HashSet store elements in random order. | LinkedHashSet is the sub class of HashSet and it store elements in the order how the elements are inserted. |

**Q) how many null values we can store in ArrayList,HashSet?**

We can add multiple null values in ArrayList. And we can add only one null value in HashSet.

**Q) how many null values we can store in TreeSet or SortedSet?**

SortedSet (or) TreeSet does not allow null value. If we add null value, it throws **NullPointerException**

**Q) how to add, delete, retrieve elements in List and Set,how to get an element from List or Set?**

**Add():**  in List we can add element in 2 types. add(E e) ,add(int index , E e) . in Set we can add element using add(E e).

**Delete():**in List we can delete element in 2 types. remove(Object e) ,remove(int index ) . in Set we can remove element using remove(Object e).

**Retrieve elements:** we can retrieve List elements in 4 types , using formal-for-loop, for-each-loop, Iterator,ListIterator. we can retrieve Set elements in 2 types , using for-each-loop, Iterator.

In List, Using **get(int index)** we can get element of given index. In Set we donot have **get().**

**Q) what happends if we add or delete an element in List while iterating elements?**

While iterating ArrayList , If we are trying to add or delete elements to the List, it throws java.util.ConcurrentModificationException

**Q) how HashSet is implemented (or) internal implementation of HashSet (or) how Set will store unique elements?**

HashSet internally follows HashMap. It means we create object for HashSet( **new HashSet();**) it will create HashMap (**new HashMap();**). when we add element **( add(E e))** in a HashSet it calls put(key,value) method of HashMap. It means the elements will be added as key inside the Map and value will be a constant. This value will be same for all keys.

**Q) How Map will Store unique keys or How get and put method of HashMap work internally".?**

Refer the below links <http://www.java2blog.com/2014/02/how-hashmap-works-in-java.html>

<http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-java-or.htmls>

HashMap has inner class Entry which impliments Map.Entry. the default size of HashMap is 16. And when it reaches 17 th element the size will be increased by 0.75.

If we add 17 element HashMap size = 16+(16\*.75)= 16+(16\*(3/4))=16+12=28.

HashMap will create HashMap$Entry table with 16 entries[0 to 15]. Each entry is called as a bucket. Each bucket has a LinkedList(it is not java.util.LinkedList).

If we put entries in HashMap, below operations will be performed.

* If we are adding **null** as key Map will store it in 0 th index of table.
* If key is not a null, HashMap will check hashCode() of the key using hash().
* After that it will calculate index of the key using hash value to store the element.
* Once we get index HashMap will check if any other key is already present in the entry or not. If the entry has already some keys, it will check each key in the bucket with equals().
* If the key is not matched with any other key it will insert at last position of LinkedList. If key is matched, it will override the entry(key,value) pair.
* There is no logic applied on value of the HashMap entries.

If we call get(Object o), below operation will be performed

* If we pass **null** as key,it will get value from 0 th index of table. Because HashMap always stores null key at 0 th index.
* If key is not a null, HashMap will check hashCode() of the key using hash().
* After that it will calculate index of the key using hash value to find where the element is stored.
* Once we get index HashMap will check if any other key is already present in the entry or not. If the entry has already some keys, it will check each key in the bucket with equals().
* If the key is not matched with any other key it will return null value. If key is matched, it will return the value .
* There is no logic applied on value of the HashMap entries.

**Q) difference ,similarity b/w Collection and Map?**

Both Collection and Map are Interfaces. There is no parent, child relationship b/w these.

|  |  |
| --- | --- |
| **Collection** | **Map** |
| It stores 1-dimentional values. [1,2,3,… etc] | It stores 2-dimentional values.[{1,2},{2,2},..etc]we will call Map elements as {key, value } pairs |

**Q)how can insert values in Map why we do not have add() in Map (or) why put() in Map , why not add() ?**

A) Collection has add(E e) for adding elements, Map has put(key,value) for adding entries. there is no parent child relation b/w Map and Collection. That is why add() is not present in Map.

**Q) explain about Map rules and methods?**

Map contains {key,value} pairs. Keys follow **Set** rules and values follow **Collection or List** rules. It allows 1 null key multiple null values. if we add duplicate key for more than one time, the latest entry overrides the previous entry. . Map always stores null key entry at 0 th index. Map has Entry as inner interface. It refers each {key, value} pair as entry.

|  |  |
| --- | --- |
| **Method** | **Description** |
| int size() | It gives size of the Map starting from 1. |
| boolean isEmpty(); | It checks whether Map is empty or not. |
| boolean containsKey(Object key); | It returns true if key is present in Map. |
| boolean containsValue(Object value); | It returns true if value is present in Map. |
| V get(Object key); | It returns value of given key, if the key is not present it returns **null.** |
| V put(K key, V value); | It is used to add {key,value} entry in Map. It returns value, |
| V remove(Object key); | It is used to remove entry of the given key . It returns value, |
| void clear(); | It is used to remove all entries from Map. |
| Set<K> keySet(); | It returns all the keys from Map as Set. |
| Collection<V> values(); | It returns all the values from Map as Collection. |
| Set<Map.Entry<K, V>> entrySet(); | It returns all the entries from Map as Set. |
| K getKey(); V getValue(); | These are the methods of Map.Entry . getKey() returns key of the entry, getValue() returns value of the entry, |

**Q) difference b/w HashMap and SortedMap?**

|  |  |
| --- | --- |
| **HashMap** | **SortedMap (or) TreeMap** |
| HashMap store entries in random order based on keys. | SortedMap store entries in sorted order(by default ascending order) based on keys. |
| HashMap allows one null key , multiple null values. | SortedMap does not allow null key . If we add null key in a entry , it will throws **NullPointerException.** It allows multiple null values |

**Q) difference b/w HashMap and LinkedHashMap?**

A) Both HashMap and LinkedHashMap allow one null key in a entry and allow multiple null values. both store unique entries.

|  |  |
| --- | --- |
| **HashMap** | **LinkedHashMap** |
| HashMap store entries in random order based on keys. | LinkedHashMap is the sub class of HashMap and it store entries in the order how the keys are inserted. |
| If a key is duplicated multiple times, HashMap will store the latest entry{key,value} | If a key is duplicated multiple times, LinkedHashMap will store the latest entry{key,value}. Here key index will not be changed. |

**Q) what is java.util.Map.Entry ?**

A) java.util.Map.Entry is a inner interface of Map. We can not access Entry directly .

**Q) how many null keys and values are allowed in HashMap ,what happens if we add more than one null key or null value?**

A) It allows 1 null key, multiple null values. if we add duplicate key for more than one time, the latest entry overrides the previous entry. Map always stores null key entry at 0 th index.

**Q) how many null keys and values are allowed in TreeMap or SortedMap, what happens if we add more than one null key or null value?**

A) SortedMap or TreeMap does not allow null key . If we add null key in a entry , it will throws **NullPointerException.** It allows multiple null values

**Q) Hahtable rules? how many null keys and values are allowed in HashTable , what happens if we add more than one null key or null value?**

A)Hashtable is a legacy collection, Hashtable is a sub class Map and it contains {key,value} pairs similar to HashMap. It means it allows unique keys and duplicate values. but Hashtable does not allow null key, null value . If we add null key or null value in a entry , it will throws **NullPointerException**. In HashTable it will check null value first, later it will check null key .many of the methods of Hashtable are synchronized. Due to this it slows the performance.

**Q) in how many ways Map can be iterated?**

A) 4 ways

{keySet() using For-each-loop, entrySet() using For-each-loop} {keySet() using Iterator, entrySet() using Iterator}

Q) what is use of java.util.Arrays class ?

Arrays is a class which contains private default constructor and static methods. So we can not create object for Arrays .

Methods 1) public static <T> List<T> asList(T... a) {

return new ArrayList<>(a);

}

asList(T… a): this method takes an array of any type(int,char,String,userdefind class ) and returns List.

public static void sort(int[] a) : sort( ) method accepts an array of any type (int,char,String,userdefind class) Arrays class has overloaded sort() methods which accepts all data types of array.

public static boolean equals(Object[] a, Object[] a2) : equals() is used to compare 2 Arrays of any type (2 int[],2 char[],2 String[],2 userdefind class []) Arrays class has overloaded sort() methods which accepts all data types of array.

Q) what is use of java.util.Collections class ?

Mostly we use Collections class for sorting the list of elements or objects. We can not create object for this because it has private default constructor. It has all static methods.

**InnerClasses / NestedClasses :**

InnerClass means defining a class in another class.

Types of InnerClasses



In a class , We can add instance class , static class , interface , interface of static type.

In a interface , We can add instance class , static class , interface , interface of static type.

**Anonymous InnerClass** is a temporary class which is created at compilation time.

we create Anonymous InnerClass for abstract class and interface.

**Comparable and Comparator :**

**Q) How can you sort collection of objects? Or How can you sort collection of objects based on any argument in the class?**

**A)** we can sort collection of objects using Comparable and Comparator interfaces

|  |  |
| --- | --- |
| **Java.lang.Comparable** | **Java.util.Comparator** |
| Comparable provides **single sorting sequence**. That means if an Employee has 3 variables(int id, String name, String desg), we can sort the list of Employees on the basis of single variable(at a time we can apply sorting only on 1 variable). If we want to sort based on 2 variables, it will not possible . | Comparator provides **multiple sorting sequence**. That means if an Employee has 3 variables(int id, String name, String desg), we can sort the list of Employees on the basis of single variable or all the variables |
| Comparable **affects the original class** i.e. Employee class must implement Comparable. | Comparator **doesn't affect the original class** i.e we don’t need to implement Comparator for Employee class or any other class. |
| Comparable provides **compareTo() method** to sort elements.  Syntax: public int compareTo(T o); | Comparator provides **compare() method** to sort elements.  Syntax: int compare(T o1, T o2); |
| Comparable is found in **java.lang** package. | Comparator is found in **java.util** package. |
| We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List,Comparator)** method.  We use anonymous inner class for Comparator interface in this implementation. |

Q) why we do not override equals() of Comparator?

Comparator has int compare(T o1, T o2) and public Boolean equals(Object o) . but equals() is java.lang.Object Class method. So if any class is implementing Comparator interface, we should override compare() and overriding equals() is optional. Because it is already inherited from Object class.

**Thread:** In java Thread means **“** a flow of control**”** or “single flow of execution**”**

Thread is a class in java.lang package. we can crate thread in 2 ways

1. By extending Thread class
2. By implementing Runnable interface.

If we extend a class C1 to Thread , the class C1 will be treated as a Thread

Example: class Multi extends Thread{

public void run(){

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

}

public static void main(String args[]){

Multi t1=new Multi();

t1.start();

}

}

In the above example class Multi is treated as Thread . Thread as a lifecycle. It has 5 stages ( new , runnable , running, waiting, terminated).

We can create a Thread by implementing java.lang.Runnable interface as below

Example: class Multi3 implements Runnable{

public void run(){

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

}

public static void main(String args[]){

Multi3 m1=new Multi3();

Thread t1 =new Thread(m1);

t1.start();

}

}

Enum is a data type and it is used for creating set of constants

Example: public enum E{

C1 ,C2 , C3, C4, C5

}

Enum can have constructors , variables and methods we can not extend enum to any class, because enum is extending Enum class internally. We can not create object for Enum.

**Java Heap space and Stack memory:**

**Jvm** is divided in to different types of memory to execute an application.

## Java Heap Space

Java Heap space is used by java runtime to allocate memory to Objects and JRE classes(like Object , String , Wrapper Classes and all Other Classes). Whenever we create any object, it’s always created in the Heap space.

### Java Stack Memory

Java Stack memory is used for execution of a method. They contain primitives which are created inside a method , object references.

Stack memory is always referenced in LIFO (Last-In-First-Out) order. Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method. As soon as method ends, the block becomes unused and become available for next method.  
Stack memory size is very less compared to Heap memory.

StringPool: it is a separate memory in side heap memory. It is used for holding the String values which are created by String literals

String s1 = “String1”;//this is form of String literal creation ,which stores in String Pool.

String s1 = new String(“String1”);// this is String object creation which stores in Heap memory.

[**http://www.journaldev.com/4098/java-heap-space-vs-stack-memory**](http://www.journaldev.com/4098/java-heap-space-vs-stack-memory)

**public class Memory {**

**public static void main(String[] args) { // Line 1**

**int i=1; // Line 2**

**Object obj = new Object(); // Line 3**

**Memory mem = new Memory(); // Line 4**

**mem.foo(obj); // Line 5**

**} // Line 9**

**private void foo(Object param) { // Line 6**

**String str = param.toString(); //// Line 7**

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

**} // Line 8**

**}**

