----------Index----------

1. Java Modifiers

1) For the top level classes which modifiers are allowed?

- The only applicable modifiers for **Top Level** classes are:
    1) Public
    2) Default
    3) Final
    4) Abstract
    5) Strictfp
- If we are using any other modifier we will get compile time error.

2) Is it possible to declare a class as static, private, and protected?

- No, we cannot declare a class with static, private, protected if we declare a class with static, private, protected it will through the compile time error. **Modifier private protected static not allowed here.**

3) What are extra modifiers applicable inner classes when compared with outer classes?

- The modifiers which are applicable for inner classes but not for outer classes are **private, protected,static**.

4) What is a final class?

Final Class:

- If a class declared as the final then we can't creates the child class that is inheritance concept is not applicable for final classes.

EXAMPLE:

Program 1:

```
final class Parent
{}
```

Program 2:

```
class child extends Parent
{}
```

OUTPUT:

Compile time error.

D:\Java>javac Parent.java

D:\Java>javac child.java

child.java:1:cannot inherit from final Parent

class child extends Parent

5) Explain the difference between final, finally and finalize?

Final:

- Final is the modifier applicable for class, methods and variables.
- If a class declared as the final then child class creation is not possible.

- If a method declared as the final then overriding of that method is not possible.
- If a variable declared as the final then reassignment is not possible.

Finally:

- It is the block always associated with try catch to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and whether handled or not handled.

Finalize:

- It is a method which should be called by garbage collector always just before destroying an object to perform cleanup activities.

Note:

- To maintain clean up code finally block is recommended over finalize() method because we can't expect exact behavior of GC.

6) Is every method present in final class is final?

- Every method present inside a final class is always final by default whether we are declaring or not.

7) Is every variable present inside a final class is final?

- Every variable present inside a final class need not be final.

Example:

```
final class parent
{
static int x=10;
static
{
x=999;
}}
```

8) What is abstract class?

Abstract class:

- For any java class if we are not allow to create an object such type of class we have to declare with abstract modifier that is for abstract class instantiation is not possible.

Example:

```
abstract class Test
{
public static void main(String args[]){
Test t=new Test();
}}
```

Output:

Compile time error.

```
D:\Java>javac Test.java
Test.java:4: Test is abstract; cannot be instantiated
Test t=new Test();
```

9) What is abstract method?

Abstract Methods:

- Even though we don't have implementation still we can declare a method with abstract modifier. That is abstract methods have only declaration but not implementation. Hence abstract method declaration should compulsory ends with semicolon.

EXAMPLE:

public abstract void methodOne(); ⟶ valid
public abstract void methodOne(){} ⟶ invalid

- Child classes are responsible to provide implementation for parent class abstract methods.

10) What is the difference between abstract class and abstract method?

- If a class contain at least on abstract method then compulsory the corresponding class should be declare with abstract modifier. Because implementation is not complete and hence we can't create object of that class.
- Even though class doesn't contain any abstract methods still we can declare the class as abstract that is an abstract class can contain zero no of abstract methods also.

Example: HttpServlet class is abstract but it doesn't contain any abstract method.

11) If a class contains at least one abstract method is it required to declare that class compulsory abstract?

- If a class contain at least on abstract method then compulsory the corresponding class should be declare with abstract modifier. Because implementation is not complete and hence we can't create object of that class.

12) If a class doesn't contain any abstract method is it possible to declare that class as abstract?

- Even though class doesn't contain any abstract methods still we can declare the class as abstract that is an abstract class can contain zero no of abstract methods also.

Example: HttpServlet class is abstract but it doesn't contain any abstract method.

13) Whenever we are extending abstract class is it compulsory required to provide implantation for every abstract method of that class?

- If a class extends any abstract class then compulsory we should provide implementation for every abstract method of the parent class otherwise we have to declare child class as abstract.

14) Is final class can contain abstract method?

- Final class cannot contain abstract methods.

15) Is abstract class can contain final methods?

- Abstract class can contain final method.

16) What is the difference between final and abstract?

- For abstract methods compulsory we should override in the child class to provide implementation. Whereas for final methods we can't override hence abstract final combination is illegal for methods.
- For abstract classes we should compulsory create child class to provide implementation whereas for final class we can't create child class. Hence final abstract combination is illegal for classes.
- Final class cannot contain abstract methods whereas abstract class can contain final method.

17) Can u give example for abstract class which doesn't contain any abstract method?

Example: HttpServlet class is abstract but it doesn't contain any abstract method.

18) Which of the following modifiers combinations are legal for methods?

1) **Public-static:** legal combination for methods.
2) **Static-abstract:** Illegal combination of modifiers: abstract and static(C.E). Because For static methods compulsory implementation should be available where as for abstract methods implementation should not be available **hence abstract static combination is illegal for methods.**
3) **Abstract-final:** Illegal combination of modifiers: abstract and final(C.E). Because abstract methods we have to override in child classes where as final methods we can't override in the child classes.
4) **Final-synchronized:** legal combination for methods.
5) **Synchronized-native:** legal combination for methods.
6) **Native-abstract:** Illegal combination of modifiers: abstract and native

19) Which of the following modifiers combinations are legal for classes?

1) **Public-final:** legal combination for classes.
2) **Abstract-final:**Illegal combination of modifiers for classes: abstract and final(C.E). Because for abstract classes we have to create child class where as for final classes we can't create child class.
3) **Abstract-strictfp:**legal combination for classes.
4) **Strictfp-public:**legal combination for classes.

20) What is strictfp modifier?

Strictfp:

- Strictfp is the modifier applicable for methods and classes but not for variables.
- Strictfp modifier introduced in 1.2 versions.
- If a method declare as the Strictfp then all the floating point calculations in that method has to follow IEEE754 standard. So that we will get flat from independent results.

Example:

```
System.out.println(10.0/3);
P4                  P3          IEEE754
3.33333333333333    3.333333    3.333
```

- If a class declares as the Strictfp then every concrete method(which has body) of that class has to follow IEEE754 standard for floating point arithmetic.

21) Is it possible to declare a variable with strictfp?

- Strictfp is the modifier applicable for methods and classes but not for variables.

22) Abstract-strictfp combination, is legal for classes (or) methods?

- Abstract strictfp combination is legal for classes but illegal for methods.

23) What is the difference between abstract and strictfp?

- Strictfp method talks about implementation where as abstract method never talks about implementation hence **abstract,strictfp** combination is illegal for methods.
- But we can declare a class with abstract and strictfp modifier simultaneously. That is abstract strictfp combination is legal for classes but illegal for methods.

Example:

```
public abstract strictfp void methodOne(); (invalid)
abstract strictfp class Test  (valid)
{
}
```

24) Is it possible to override a native method?

- For native methods inheritance, overriding and overloading concepts are applicable.

25) What is the difference between instance and static variable?

Instance variables:

- If the value of a variable is varied from object to object such type of variables are called instance variables.
- For every object a separate copy of instance variables will be created.
- Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly same as scope of objects.
- Instance variables will be stored on the heap as the part of object.
- Instance variables should be declared with in the class directly but outside of any method or block or constructor.
- Instance variables can be accessed directly from Instance area. But cannot be accessed directly from static area.
- But by using object reference we can access instance variables from static area.

Static variables:

- If the value of a variable is not varied from object to object such type of variables is not recommended to declare as instance variables. We have to declare such type of variables at class level by using static modifier.
- In the case of instance variables for every object a separate copy will be created but in the case of static variables for entire class only one copy will be created and shared by every object of that class.
- Static variables will be crated at the time of class loading and destroyed at the time of class unloading hence the scope of the static variable is exactly same as the scope of the **.class** file.
- Static variables will be stored in method area. Static variables should be declared with in the class directly but outside of any method or block or constructor.
- Static variables can be accessed from both instance and static areas directly.
- We can access static variables either by class name or by object reference but usage of class name is recommended.
- But within the same class it is not required to use class name we can access directly.

26) What is the difference between general static variable and final static variable?

Final static variables:

- If the value of a variable is not varied from object to object such type of variables is not recommended to declare as the instance variables. **We have to declare those variables at class level by using static modifier.**
- For the static variables it is not required to perform initialization explicitly jvm will always provide default values.

Example:

```
class Test
{
static int i;
public static void main(String args[]){
System.out.println("value of i is :"+i);
}}
```

Output:

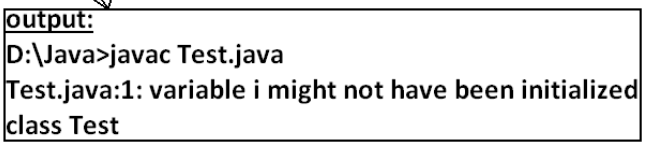
```
D:\Java>javac Test.java
D:\Java>java Test
Value of i is: 0
```

- If the static variable declare as final then compulsory we should perform initialization explicitly whether we are using or not otherwise we will get compile time error.

Example:

```
class Test
{
static int i;
}
```
valid

```
class Test
{
final static int i;
}
```
invalid



```
output:
D:\Java>javac Test.java
Test.java:1: variable i might not have been initialized
class Test
```

27) Which modifiers are applicable for local variable?

- The only applicable modifier for local variables is final if we are using any other modifier we will get compile time error.

28) When the static variables will be created?

- Static variables will be crated at the time of class loading and destroyed at the time of class unloading hence the scope of the static variable is exactly same as the scope of the **.class** file.

29) What are various memory locations of instances variables, local variables and static variables?

- Instance variables will be stored on the heap as the part of object.
- Static variables will be stored in method area.
- The local variables will be stored on the stack.

30) Is it possible to overload a main method?

- Overloading of the main() method is possible but JVM always calls string[] argument main() method only.
- The other overloaded method we have to call explicitly then only it will be executed.

31) Is it possible to override static methods?

Case 1:

- We can't override a static method as non static.

Example:

```
class Parent
{
        public static void methodOne()//here static methodOne() method is a class level
        {}
}
class Child extends Parent
{
        public void methodOne()//heremethodOne() method is a object level hence we can'toverride methodOne() method
```

```
    {}
}
```

Case 2:

- Similarly we can't override a non static method as static.

Case 3:

```
class Parent
{
public static void methodOne()
{}
}
class Child extends Parent
{
public static void methodOne()
{}
}
```

- It is valid. It seems to be overriding concept is applicable for static methods but it is not overriding it is met1hod hiding.

32) What is native keyword & where it is applicable?

Native modifier:

- Native is a modifier applicable only for methods but not for variables and classes.
- The methods which are implemented in non java are called native methods or foreign methods.

33) What is the main advantage of the native keyword?

The main objectives of native keyword are:

- To improve performance of the system.
- To use already existing legacy non java code.

34) If we are using native modifier how we can maintain platform independent nature?

- The main disadvantage of native keyword is usage of native keyword in java breaks platform independent nature of java language.

35) How we can declare a native method?

- For native methods implementation is already available and we are not responsible to provide implementation hence native method declaration should compulsory ends with semicolon.
    - Public native void methodOne()----invalid
    - Public native void methodOne();---valid

36) Is abstract method can contain body?

- Abstract method cannot have a body.

37) What is synchronized keyword where we can apply?

- Synchronized is the modifier applicable for methods and blocks but not for variables and classes.
- If a method or block declared with synchronized keyword then at a time only one thread is allow to execute that method or block on the given object.

38) What are advantages & disadvantages of synchronized keyword?

- The main advantage of synchronized keyword is we can resolve data inconsistency problems, but the main disadvantage is it increases waiting time of the threads and effects performance of the system. Hence if there is no specific requirement never recommended to use synchronized keyword.

39) Which modifiers are the most dangerous in java?

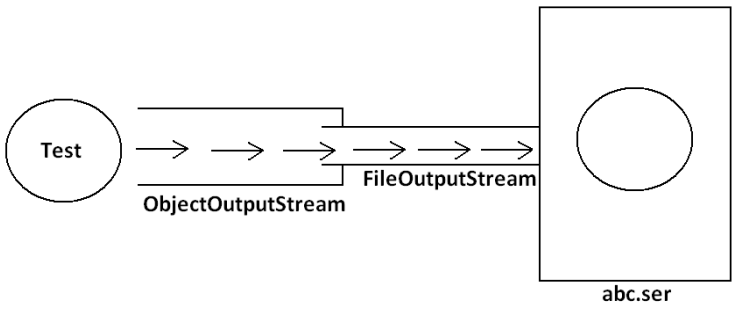
1) **Synchronized:**The main disadvantage is it increases waiting time of the threads and effects performance of the system. Hence if there is no specific requirement never recommended to use synchronized keyword.
2) **Native:**The main disadvantage of native keyword is usage of native keyword in java breaks platform independent nature of java language.
3) **Final:**The main disadvantage is we are missing the key benefits of oops: polymorsim (because of final methods), inheritance (because of final classes) hence if there is no specific requirement never recommended to use final keyboard.
4) **Volatile:**The main advantage of volatile modifier is we can resolve data inconsistency problems, but creating and maintaining a separate copy for every thread increases complexity of the programming and effects performance of the system. Hence if there is no specific requirement never recommended to use volatile modifier and it's almost outdated.

40) What is serialization & explain how its process?

Serialization: The process of saving (or) writing state of an object to a file is called serialization but strictly speaking it is the process of converting an object from java supported form to either network supported form (or) file supported form.

- By using FileOutputStream and ObjectOutputStream classes we can achieve serialization process.

Diagram:

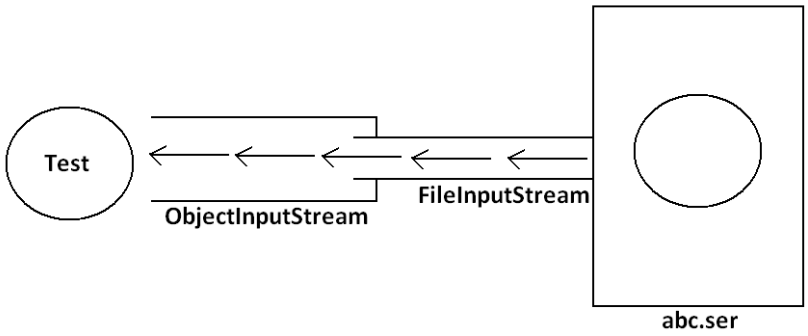


41) What is de-serialization?

De-Serialization: The process of reading state of an object from a file is called DeSerialization but strictly speaking it is the process of converting an object from file supported form (or) network supported form to java supported form.

- By using FileInputStream and ObjectInputStream classes we can achieve DeSerialization.

Diagram:



Example:

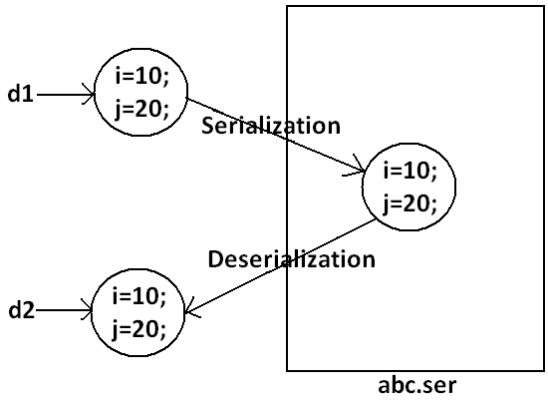
```
import java.io.*;
class Dog implements Serializable
{
int i=10;
int j=20;
}
class SerializableDemo
{
public static void main(String args[])throws Exception{
Dog d1=new Dog();
System.out.println("Serialization started");
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(d1);
System.out.println("Serialization ended");
System.out.println("Deserialization started");
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Dog d2=(Dog)ois.readObject();
System.out.println("Deserialization ended");
System.out.println(d2.i+"....."+d2.j);}}
```

Output:

```
Serialization started
```

```
Serialization ended
Deserialization started
Deserialization ended
10.................20
```

Diagram:



42) By using which classes we can achieve serialization & de-serialization?

- By using FileOutputStream and ObjectOutputStream classes we can achieve serialization process.
- By using FileInputStream and ObjectInputStream classes we can achieve DeSerialization.

43) What is Serializable interface and explain its methods?

- Serializable interface present in **java.io package** and does not contain any methods. It is marker interface. The required ability will be provided automatically by JVM.

44) Without having any method in Serializable interface, how we can get Serializable ability for our object?

- The required ability will be provided automatically by JVM.

45) What is the purpose of transient keyword and explain its advantages?

Transient keyword:

- While performing serialization if we don't want to serialize the value of a particular variable then we should declare that variable with "transient" keyword.
- At the time of serialization JVM ignores the original value of transient variable and save default value.
- That is transient means "not to serialize".

46) Is it possible to serialize every java object?

- We can perform Serialization only for Serializable objects.
- An object is said to be Serializable if and only if the corresponding class implements Serializable interface.

47) Is it possible to declare a method a class with transient?

- Transient is the modifier applicable only for variables but not for methods and classes.

48) If we declare static variable with transient is there any impact?

- Static variable is not part of object state hence they won't participate in serialization because of this declaring a static variable as transient there is no use.

49) What is the impact of declaring final variable a transient?

- Final variables will be participated into serialization directly by their values. Hence declaring a final variable as transient there is no use.

50) What is volatile variable?

- Volatile is the modifier applicable only for variables but not for classes and methods.
- If the value of variable keeps on changing such type of variables we have to declare with volatile modifier.
- If a variable declared as volatile then for every thread a separate local copy will be created by the jvm, all intermediate modifications performed by the thread will takes place in the local copy instead of master copy.
- Once the value got finalized before terminating the thread that final value will be updated in master copy.

51) Is it possible to declare a class or method with volatile?

- Volatile is the modifier applicable only for variables but not for classes and methods.

52) What is the advantage and disadvantage of volatile modifier?

- The main advantage of volatile modifier is we can resolve data inconsistency problems, but creating and maintaining a separate copy for every thread increases complexity of the programming and effects performance of the system. Hence if there is no specific requirement never recommended to use volatile modifier and it's almost outdated.

Final Modifier:

- Final is the modifier applicable for classes, methods and variables.
- The main advantage of final keyword is we can achieve security. Whereas the main disadvantage is we are missing the key benefits of oops: polymorphism (because of final methods), inheritance (because of final classes) hence if there is no specific requirement never recommended to use final keyboard.

Abstract Modifier:

- Abstract is the modifier applicable only for methods and classes but not for variables.

Abstract Methods:

- Even though we don't have implementation still we can declare a method with abstract modifier. That is abstract methods have only declaration but not implementation. Hence abstract method declaration should compulsory ends with semicolon.

EXAMPLE:

```
public abstract void methodOne(); ------> valid
public abstract void methodOne(){} ------> invalid
```

- Child classes are responsible to provide implementation for parent class abstract methods.
- The main advantage of abstract methods is , by declaring abstract method in parent class we can provide guide lines to the child class such that which methods they should compulsory implement.
- Abstract method never talks about implementation whereas if any modifier talks about implementation it is always illegal combination.

Abstract class:

- For any java class if we are not allow to create an object such type of class we have to declare with abstract modifier that is for abstract class instantiation is not possible.

Static modifier:

- Static is the modifier applicable for methods, variables and blocks.
- We can't declare a class with static **but inner classescan be declaring as the static**.
- In the case of instance variables for every object a separate copy will be created but in the case of static variables a single copy will be created at class level and shared by all objects of that class.

Native modifier:

- Native is a modifier applicable only for methods but not for variables and classes.
- The methods which are implemented in non java are called native methods or foreign methods.

The main objectives of native keyword are:

- To improve performance of the system.
- To use already existing legacy non java code.
- The main disadvantage of native keyword is usage of native keyword in java breaks platform independent nature of java language.

Synchronized:

- Synchronized is the modifier applicable for methods and blocks but not for variables and classes.
- If a method or block declared with synchronized keyword then at a time only one thread is allow to execute that method or block on the given object.
- The main advantage of synchronized keyword is we can resolve data inconsistency problems, but the main disadvantage is it increases waiting time of the threads and effects performance of the system. Hence if there is no specific requirement never recommended to use synchronized keyword.

Transient modifier:

- Transient is the modifier applicable only for variables but not for methods and classes.
- At the time of serialization if we don't want to serialize the value of a particular variable to meet the security constraints then we should declare that variable with transient modifier.
- At the time of serialization jvm ignores the original value of the transient variable and save default value that is transient means "not to serialize".
- Static variables are not part of object state hence serialization concept is not applicable for static variables duo to this declaring a static variable as transient there is no use.
- Final variables will be participated into serialization directly by their values due to this declaring a final variable as transient there is no impact.

Volatile modifier:

- Volatile is the modifier applicable only for variables but not for classes and methods.
- If the value of variable keeps on changing such type of variables we have to declare with volatile modifier.
- The main advantage of volatile modifier is we can resolve data inconsistency problems, but creating and maintaining a separate copy for every thread increases complexity of the programming and effects performance of the system. Hence if there is no specific requirement never recommended to use volatile modifier and it's almost outdated.

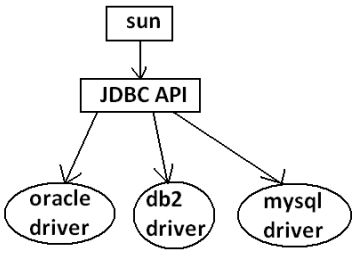
Interfaces

1) What is interface?

Def1: Any service requirement specification (srs) is called an interface.

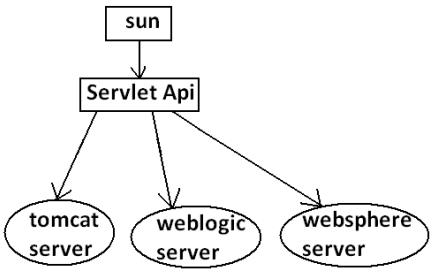
Example1: Sun people responsible to define JDBC API and database vendor will provide implementation for that.

Diagram:



Example2: Sun people define SERVLET API to develop web applications web server vendor is responsible to provide implementation.

Diagram:



Def2: From the client point of view an interface define the set of services what he is excepting. From the service provider point of view an interface defines the set of services what is offering. Hence an interface is considered as a contract between client and service provider.

Example: ATM GUI screen describes the set of services what bank people offering, at the same time the same GUI screen the set of services what customer his excepting hence this GUI screen acts as a contract between bank and customer.

Def3: Inside interface every method is always abstract whether we are declaring or not hence interface is considered as 100% pure abstract class.

Summery def: Any service requirement specification (SRS) or any contract between client and service provider or 100% pure abstract classes is considered as an interface.

2) What is difference between interface and abstract class?

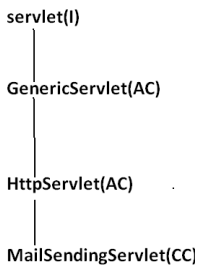
Interface	Abstract class
1) If we don't' know anything about implementation just we have requirement specification then we should go for interface.	1) If we are talking about implementation but not completely (partial implementation) then we should go for abstract class.

2) Every method present inside interface is always **publicand abstract** whether we are declaring or not.	2) Every method present inside abstract class **need not be public and abstract**.
3) We can't declare interface methods with the modifiers **private**, **protected**, **final**, **static**, **synchronized**, **native**, **strictfp**.	3) There are no restrictions on abstract class method modifiers.
4) Every interface variable is always **public static final** whether we are declaring or not.	4) Every abstract class variable need not be public static final.
5) Every interface variable is always **public staticfinal** we can't declare with the following modifiers. **Private, protected, transient, volatile.**	5) There are no restrictions on abstract class variable modifiers.
6) For the interface variables compulsory we should perform initialization at the time of declaration otherwise we will get compile time error.	6) It is not require to perform initialization for abstract class variables at the time of declaration.
7) Inside interface we can't take static and instance blocks.	7) Inside abstract class we can take both static and instance blocks.
8) Inside interface we can't take constructor.	8) Inside abstract class we can take constructor.

3) When we should go for interface and abstract class and concrete class?

- If we don't know anything about implementation just we have requirement specification then we should go for interface.
- If we are talking about implementation but not completely (partial implementation) then we should go for abstract class.
- If we are talking about implementation completely and ready to provide service then we should go for concrete class.

Example:



4) What modifiers applicable for interfaces?

1) Public
2) Abstract
3) Strictfp

5) Explain about interface variables and what modifiers are applicable for them?

Interface variables:

- An interface can contain variables to define requirement level constants.
- Every interface variable is always **public static final** whether we are declaring or not.
- Every interface variable is always **public staticfinal** we can't declare with the following modifiers. **Private, protected, transient, volatile.**
- For the interface variables compulsory we should perform initialization at the time of declaration otherwise we will get compile time error.

Example:

```
interface interf
{
int x=10;
}
```

Public: To make it available for every implementation class.

Static: Without existing object also we have to access this variable.

Final: Implementation class can access this value but cannot modify.

6) Explain about interface methods and what modifiers are applicable for them?

- Every method present inside interface is always **publicand abstract** whether we are declaring or not.
- We can't declare interface methods with the modifiers **private**, **protected**, **final**, **static**, **synchronized**, **native**, **strictfp**.

7) Can java class implement any number of interfaces?

- A class can implements any no. Of interfaces at a time.
- Except if two interfaces contains a method with same signature but different return types.

Example:

```
interface One{
public void methodOne();
}
interface Two{
public void methodTwo();
}
class Three implements One,Two{
public void methodOne(){
}
```

```
public void methodTwo(){
}}
```

8) If two interfaces contains a method with same signature but different return types, then how we can implement both interfaces simultaneously?

- If two interfaces contain a method with same signature but different return types then it is not possible to implement both interfaces simultaneously.

Example 1:

```
interface Left
{
public void methodOne();
}
```

Example 2:

```
interface Right
{
public int methodOne(int i);
}
```

- We can't write any java class that implements both interfaces simultaneously.

9) Difference between extends and implements keyword?

- A class can extend only one class at a time.
- A class can implements any no. Of interfaces at a time.
- A class can extend a class and can implement an interface simultaneously.
- An interface can extend any no. Of interfaces at a time.

10) We cannot create an object of abstract class then what is necessity of having constructor inside abstract class?

- This constructor will be executed for the initialization of child object.

11) What is a marker interface? Give an example?

Marker interface: If an interface doesn't contain any methods and by implementing that interface if our object gets some ability such type of interfaces are called Marker interface (or) Tag interface (or) Ability interface.

Example:

Serializable
Cloneable
RandomAccess
SingleThreadModel
} These are marked for some ability

12) Without having any methods in marker interface how objects will get ability?

- Internally JVM will provide required ability.

13) Why JVM is providing the required ability?

- To reduce complexity of the programming.

14) Is it possible to create our own marker interface?

- Yes, but customization of JVM is required.

15) What is adapter class and explain its usage?

Adapter class:

- Adapter class is a simple java class that implements an interface only with empty implementation for every method.
- If we implement an interface directly for each and every method compulsory we should provide implementation whether it is required or not. This approach increases length of the code and reduces readability.

Example 1:

```
interface X{
void m1();
void m2();
void m3();
void m4();
 //.
 //.
void m5();
}
```

Example 2:

```
class Test implements X{
public void m3(){
System.out.println("m3() method is called");
}
public void m1(){}
public void m2(){}
public void m4(){}
public void m5(){}
}
```

- We can resolve this problem by using adapter class.
- Instead of implementing an interface if we can extend adapter class we have to provide implementation only for required methods but not for all methods of that interface.
- This approach **decreases length of the code** and improves readability.
- Generic Servlet simply acts as an adapter class for Servlet interface.

16) An interface contains only abstract methods and an abstract class also can contain only abstract methods then what is the necessity of interface?

- We can replace interface concept with abstract class. But it is not a good programming practice. We are misusing the roll of abstract class.

17) In your previous project where you used the following marker interface, abstract class, interface, adapter class?

OOPS

1) What is data hiding?

Data Hiding:

- Our internal data should not go out directly that is outside person can't access our internal data directly.
- By using private modifier we can implement data hiding.

Example:

```
class Account
{
        private double balance;
        .....;
        .....;
}
```

- The main advantage of data hiding is security.

Note: Recommended modifier for data members is private.

2) What is abstraction?

Abstraction:

- Hide internal implementation and just highlight the set of services, is called abstraction.
- By using abstract classes and interfaces we can implement abstraction.

Example:

- By using ATM GUI screen bank people are highlighting the set of services what they are offering without highlighting internal implementation.
- The main advantages of Abstraction are:

1) We can achieve security as we are not highlighting our internal implementation.
2) Enha3
3) ncement will become very easy because without effecting end user we can able to perform any type of changes in our internal system.
4) It provides more flexibility to the end user to use system very easily.
5) It improves maintainability of the application.

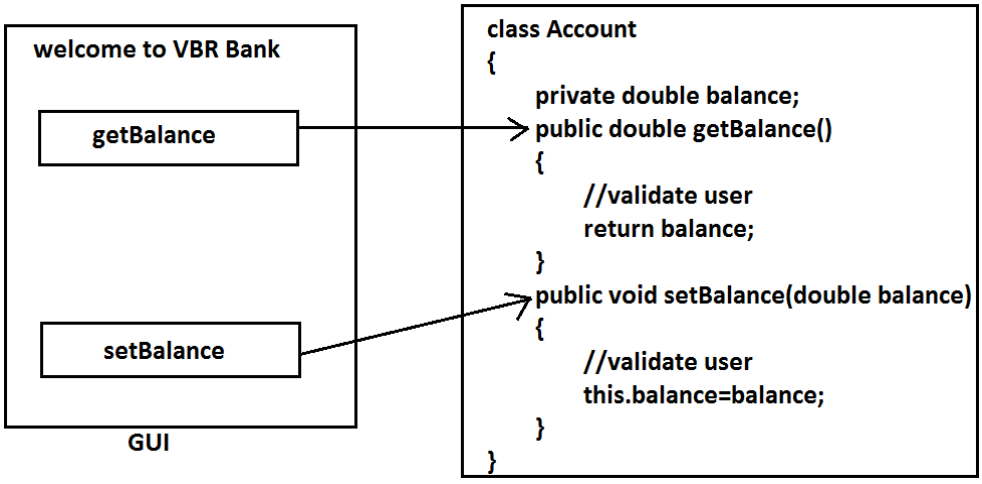
3) What is encapsulation?

Encapsulation:

- It is the process of Encapsulating data and corresponding methods into a single module.
- If any java class follows data hiding and abstraction such type of class is said to be encapsulated class.

Encapsulation=Datahiding+Abstraction

Example:



- In encapsulated class we have to maintain getter and setter methods for every data member.

4) Explain the advantages and disadvantages of encapsulation?

- The main advantages of encapsulation are:

1) We can achieve security.
2) Enhancement will become very easy.
3) It improves maintainability of the application.
4) It provides flexibility to the user to use system very easily.

- The main disadvantage of encapsulation is it increases length of the code and slows down execution.

5) What is tightly encapsulated class?

Tightly encapsulated class:

- A class is said to be tightly encapsulated if and only if every variable of that class declared as private whether the variable has getter and setter methods are not and whether these methods declared as public or not, not required to check.

Example:

```
class Account
{
        private double balance;
        public double getBalance()
        {
                return balance;
        }
}
```

6) What is IS-A relationship?

IS-A Relationship(inheritance):

1) Also known as inheritance.
2) By using extends keywords we can implement IS-A relationship.
3) The main advantage of IS-A relationship is reusability.

Example:

```
class Parent
{
        public void methodOne()
        {}
}
class Child extends Parent
{
        public void methodTwo()
        {}
}
class Test
{
    public static void main(String[] args)
    {
        Parent p=new Parent();
        p.methodOne();
        p.methodTwo();
        Child c=new Child();
        c.methodOne();
        c.methodTwo();
        Parent p1=new Child();
        p1.methodOne();
        p1.methodTwo();
        Child c1=new Parent();
    }
}
```

C.E: cannot find symbol
symbol : method methodTwo()
location: class Parent

C.E: incompatible types
found : Parent
required: Child

Conclusion:

1) Whatever the parent has by default available to the child but whatever the child has by default not available to the parent. Hence on the child reference we can call both parent and child class methods. But on the parent reference we can call only methods available in the parent class and we can't call child specific methods.
2) Parent class reference can be used to hold child class object but by using that reference we can call only methods available in parent class and child specific methods we can't call.
3) Child class reference cannot be used to hold parent class object.

7) Which keyword is used for IS-A relationship?

- By using extends keywords we can implement IS-A relationship.

8) What are various advantages of IS-A relationship?

- The main advantage of IS-A relationship is reusability.

9) Explain about HAS-A relationship?

HAS-A relationship:

1) HAS-A relationship is also known as composition (or) aggregation.
2) There is no specific keyword to implement hAS-A relationship but mostly we can use new operator.
3) The main advantage of HAS-A relationship is reusability.

Example:

```
class Engine
{
        //engine specific functionality
}
class Car
{
        Engine e=new Engine();
        //.....;
        //.....;
        //.....;
}
```

- Class Car HAS-A engine reference.
- HAS-A relationship increases dependency between the components and creates maintains problems.

10) What is composition?

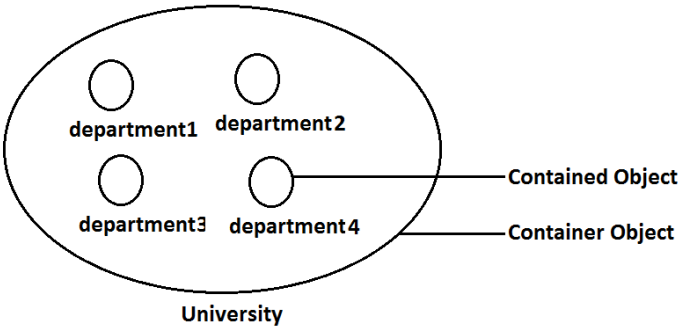
Composition:

- Without existing container object if there is no chance of existing contained objects then the relationship between container object and contained object is called composition which is a strong association.

Example:

- University consists of several departments whenever university object destroys automatically all the department objects will be destroyed that is without existing university object there is no chance of existing dependent object hence these are strongly associated and this relationship is called composition.

Diagram:



11) What is aggregation?

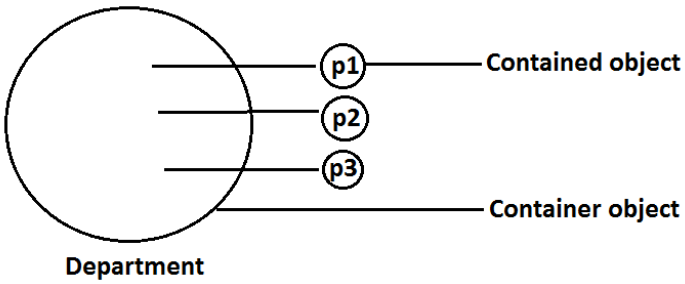
Aggregation:

- Without existing container object if there is a chance of existing contained objects such type of relationship is called aggregation. In aggregation objects have weak association.

Example:

- Within a department there may be a chance of several professors will work whenever we are closing department still there may be a chance of existing professor object without existing department object the relationship between department and professor is called aggregation where the objects having weak association.

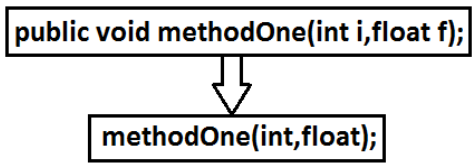
Diagram:



12) What is method signature?

Method signature: In java method signature consists of name of the method followed by argument types.

Example:



- In java return type is not part of the method signature.
- Compiler will use method signature while resolving method calls.
- Within the same class we can't take 2 methods with the same signature otherwise we will get compile time error.

Example:

```
public void methodOne()
```

```
{}
public int methodOne()
{
        return 10;
}
```

Output:

```
Compile time error
methodOne() is already defined in Test
```

13) What is overloading?

- Two methods are said to be overload if and only if both having the same name but different argument types.(or)
- Having the same name and different argument types is called method overloading.

Example:

```
class Test
{
        public void methodOne()
        {
                System.out.println("no-arg method");
        }
        public void methodOne(int i)
        {
                System.out.println("int-arg method");
        }
        public void methodOne(double d)
        {
                System.out.println("double-arg method");
        }
        public static void main(String[] args)
        {
                Test t=new Test();
                t.methodOne();//no-arg method
                t.methodOne(10);//int-arg method
                t.methodOne(10.5);//double-arg method
        }
}
```

overloaded methods

14) What are various rules satisfied by overloaded methods?

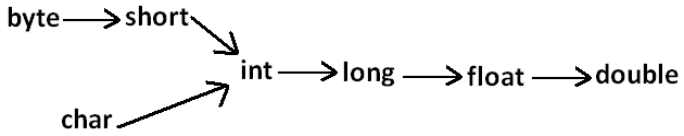
Case 1: Automatic promotion in overloading.

- In overloading if compiler is unable to find the method with exact match we won't get any compile time error immediately.
- 1st compiler promotes the argument to the next level and checks whether the matched method is available or not if it is available then that method will be considered if it is not available then compiler promotes the argument once again to the next level. This

process will be continued until all possible promotions still if the matched method is not available then we will get compile time error. This process is called automatic promotion in overloading.

- The following are various possible automatic promotions in overloading.

Diagram:



Example:

```
class Test
{
        public void methodOne(int i)
        {
                System.out.println("int-arg method");
        }
        public void methodOne(float f)
        {
                System.out.println("float-arg method");
        }
        public static void main(String[] args)
        {
                Test t=new Test();
                //t.methodOne('a');//int-arg method
                //t.methodOne(10l);//float-arg method
                t.methodOne(10.5);//C.E:cannot find symbol
        }
}
```

overloaded methods

Case 2:

```
class Test
{
        public void methodOne(String s)
        {
                System.out.println("String version");
        }
        public void methodOne(Object o)
        {
        {
```

Both methods are said to Be overloaded methods.

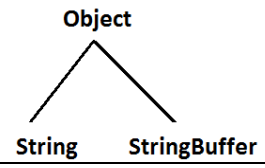
```
                System.out.println("Object version");
        }
        public static void main(String[] args)
        {
                Test t=new Test();
                t.methodOne("bhaskar");//String version
                t.methodOne(new Object());//Object version
                t.methodOne(null);//String version
        }
}
```

- In overloading Child will always get high priority then Parent.

Case 3:

```
class Test
{
        public void methodOne(String s)
        {
                System.out.println("String version");
        }
        public void methodOne(StringBuffer s)
        {
                System.out.println("StringBuffer version");
        }
        public static void main(String[] args)
        {
                Test t=new Test();
                t.methodOne("durga");//String version
                t.methodOne(new StringBuffer("bhaskar"));//StringBuffer version
        }
}
```

Output:



Case 4:

```
class Test
{
        public void methodOne(int i,float f)
        {
                System.out.println("int-float method");
        }
        public void methodOne(float f,int i)
        {
```

```
		System.out.println("float-int method");
	}
	public static void main(String[] args)
	{
		Test t=new Test();
		t.methodOne(10,10.5f);//int-float method
		t.methodOne(10.5f,10);//float-int method
		t.methodOne(10,10);//C.E:reference to methodOne is ambiguous, both
method methodOne(int,float) in Test and method methodOne(float,int) in Test match
		t.methodOne(10.5f,10.5f);//C.E:cannot find symbol
	}
}
```

Case 5:

```
class Test
{
	public void methodOne(int i)
	{
		System.out.println("general method");
	}
	public void methodOne(int...i)
	{
		System.out.println("var-arg method");
	}
	public static void main(String[] args)
	{
		Test t=new Test();
		t.methodOne();//var-arg method
		t.methodOne(10,20);//var-arg method
		t.methodOne(10);//general method
	}
}
```

- In general var-arg method will get less priority that is if no other method matched then only var-arg method will get chance for execution it is almost same as default case inside switch.

Case 6:

```
class Animal{}
class Monkey extends Animal{}
class Test
{
	public void methodOne(Animal a)
	{
		System.out.println("Animal version");
	}
```

```
        public void methodOne(Monkey m)
        {
                System.out.println("Monkey version");
        }
        public static void main(String[] args)
        {
                Test t=new Test();
                Animal a=new Animal();
                t.methodOne(a);//Animal version
                Monkey m=new Monkey();
                t.methodOne(m);//Monkey version
                Animal a1=new Monkey();
                t.methodOne(a1);//Animal version
        }
}
```

- In overloading method resolution is always based on reference type and runtime object won't play any role in overloading.

15) What is static polymorphism (or) compile-time polymorphism (or) early binding?

- In overloading compiler is responsible to perform method resolution(decision) based on the reference type. Hence overloading is also considered as compile time polymorphism(or) staticpolymorphism (or)early biding.

16) Differentiate IS-A and HAS-A relationships?

IS-A	HAS-A
1) Also known as inheritance.	1) HAS-A relationship is also known as composition (or) aggregation.
2) By using extends keywords we can implement IS-A relationship.	2) There is no specific keyword to implement hAS-A relationship but mostly we can use new operator.
3) The main advantage of IS-A relationship is reusability.	3) The main advantage of HAS-A relationship is reusability.

17) Is it possible to overload main() method?

- Overloading of the main() method is possible but JVM always calls string[] argument main() method only.
- The other overloaded method we have to call explicitly then only it will be executed.

18) In overloading whether JVM (or) compiler is responsible for method resolution?

- In overloading compiler is responsible to perform method resolution(decision) based on the reference type. Hence overloading is also considered as compile time polymorphism(or) staticpolymorphism (or)early biding.

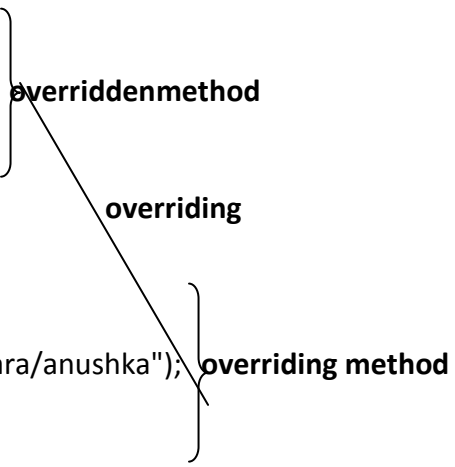
19) What is overriding?

Overriding:

- Whatever the Parent has by default available to the Child through inheritance, if the Child is not satisfied with Parent class method implementation then Child is allow to redefine that Parent class method in Child class in its own way this process is called overriding.
- The Parent class method which is overridden is called overridden method.
- The Child class method which is overriding is called overriding method.

Example:

```
class Parent
{
        public void property()
        {
                System.out.println("cash+land+gold");
        }
        public void marry()
        {
                System.out.println("subbalakshmi");
        }
}
class Child extends Parent
{
        public void marry()
        {
                System.out.println("Trisha/nayanatara/anushka");
        }
}
class Test
{
        public static void main(String[] args)
        {
                Parent p=new Parent();
                p.marry();//subbalakshmi(parent method)
                Child c=new Child();
                c.marry();//Trisha/nayanatara/anushka(child method)
                Parent p1=new Child();
                p1.marry();//Trisha/nayanatara/anushka(child method)
        }
}
```



20) What is runtime polymorphism (or) dynamic polymorphism (or) late binding?

- In overriding method resolution is always takes care by JVM based on runtime object hence overriding is also considered as runtime polymorphism (or) dynamic polymorphism (or) late binding.

21) What are various rules satisfied by overriding methods?

Rules for overriding:

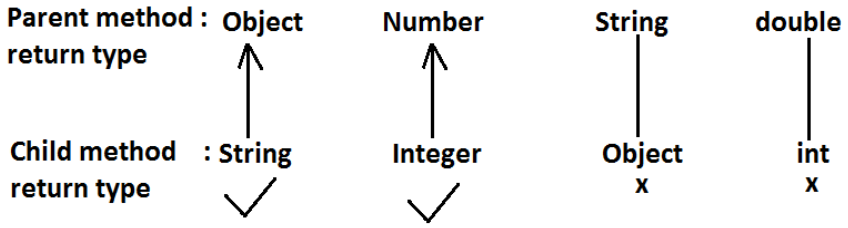
- In overriding method names and arguments must be same. That is method signature must be same.
- Until 1.4 versions the return types must be same but from 1.5 versions onwards co-variant return types are allowed.
- According to this Child class method return type need not be same as Parent class method return type its Child types also allowed.

Example:

```
class Parent
{
        public Object methodOne()
        {
                return null;
        }
}
class Child extends Parent
{
        public String methodOne()
        {
                return null;
        }
}
```

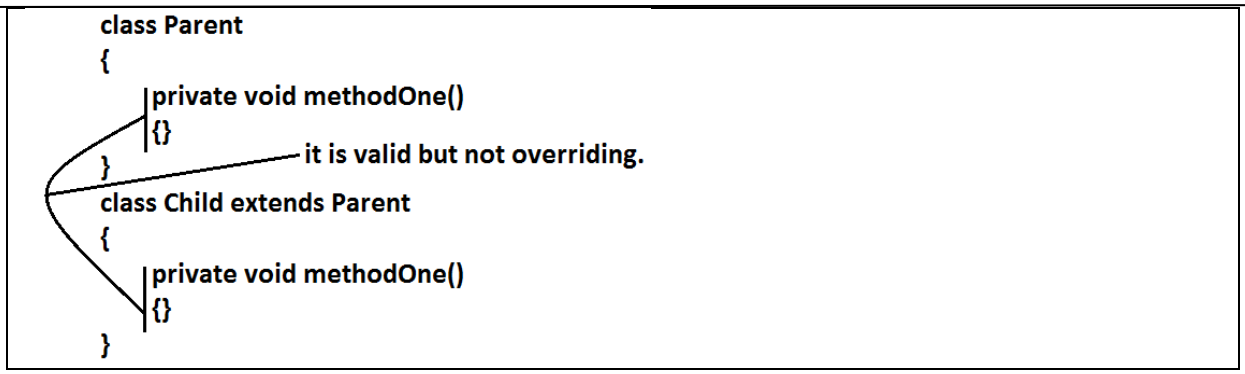
- It is valid in "1.5" but invalid in "1.4".

Diagram:



- Co-variant return type concept is applicable only for object types but not for primitives.
- Private methods are not visible in the Child classes hence overriding concept is not applicable for private methods. Based on own requirement we can declare the same Parent class private method in child class also. It is valid but not overriding.

Example:



- Parent class final methods we can't override in the Child class.

Example:

```
class Parent
{
        public final void methodOne()
        {}
}
class Child extends Parent
{
        public void methodOne()
        {}
}
```

Output:

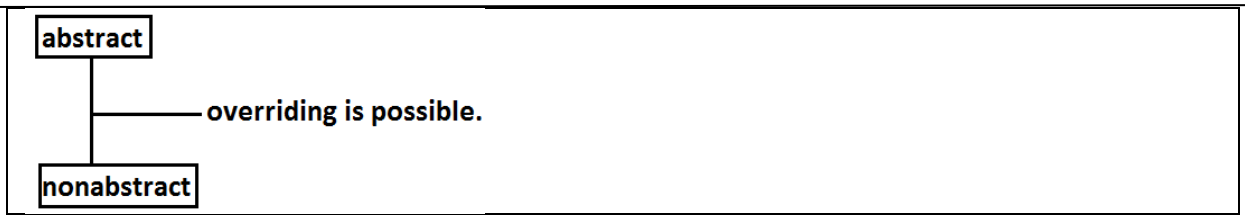
Compile time error.
Child.java:8: methodOne() in Child cannot override methodOne() in Parent; overridden method is final

- Parent class non final methods we can override as final in child class. We can override native methods in the child classes.
- We should override Parent class abstract methods in Child classes to provide implementation.

Example:

```
abstract class Parent
{
        public abstract void methodOne();
}
class Child extends Parent
{
        public void methodOne()
        {}
}
```

Diagram:



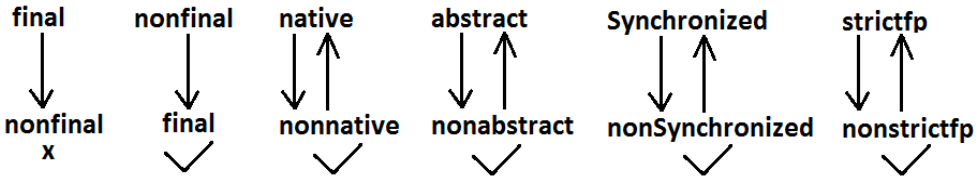
- We can override Parent class non abstract method as abstract to stop availability of Parent class method implementation to the Child classes.

Example:

```
class Parent
{
public void methodOne()
{}
}
abstract class Child extends Parent
{
public abstract void methodOne();
}
```

- Synchronized, strictfp, modifiers won't keep any restrictions on overriding.

Diagram:



- While overriding we can't reduce the scope of access modifier.

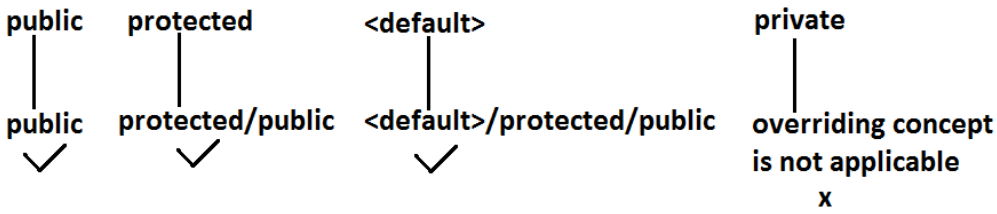
Example:

```
class Parent
{
public void methodOne()
{}
}
class Child extends Parent
{
protected void methodOne()
{}
}
```

Output:

Compile time error
methodOne() in Child cannot override methodOne() in Parent; attempting to assign weaker access privileges; was public

Diagram:



22) Is it possible to override the following methods?

Final methods: We cannot override parent class final methods.

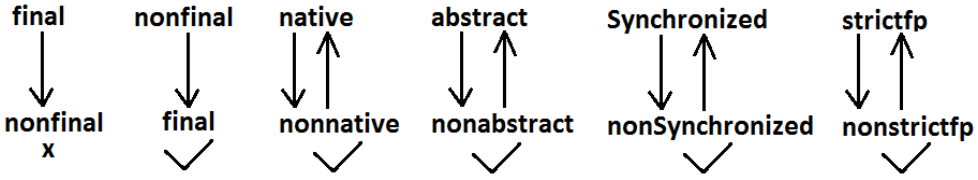
Abstract methods: We should override Parent class abstract methods in Child classes to provide implementation.

Synchronized methods: We can override synchronized methods.

Native methods: We can override native methods.

Strictfp methods: We can override Strictfp methods.

Private methods: Private methods are not visible in the Child classes hence overriding concept is not applicable for private methods



- While overriding we can't reduce the scope of access modifier.

23) What are co-variant return types?

- In overriding method names and arguments must be same. That is method signature must be same.
- Until 1.4 versions the return types must be same but from 1.5 versions onwards co-variant return types are allowed.
- According to this Child class method return type need not be same as Parent class method return type its Child types also allowed.

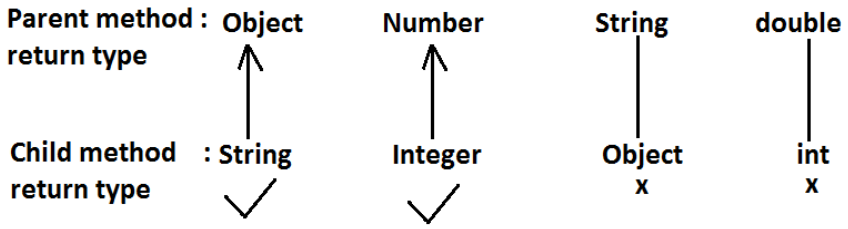
Example:

```
class Parent
{
        public Object methodOne()
        {
                return null;
        }
}
class Child extends Parent
{
        public String methodOne()
        {
                return null;
```

```
    }
}
```

- It is valid in “1.5” but invalid in “1.4”.

Diagram:



- Co-variant return type concept is applicable only for object types but not for primitives.
- Private methods are not visible in the Child classes hence overriding concept is not applicable for private methods. Based on our requirement we can declare the same Parent class private method in child class also. It is valid but not overriding.

24) While overriding is it compulsory to maintain same return type?

- In overriding method names and arguments must be same. That is method signature must be same.
- Until 1.4 version the return types must be same but from 1.5 version onwards co-variant return types are allowed.
- According to this Child class method return type need not be same as Parent class method return type its Child types also allowed.

25) In overriding who is responsible for method resolution?

- In overriding method resolution is always takes care by JVM based on runtime object hence overriding is also considered as runtime polymorphism (or) dynamic polymorphism (or) late binding.

Note: In overriding runtime object will play the role and reference type is dummy.

26) What is dynamic method dispatch?

- The process of overriding method resolution is also known as dynamic method dispatch.

27) If parent class method throws some checked exception while overriding is it compulsory should throw child class method the same checked exception?

- While overriding if the child class method throws any checked exception compulsory the parent class method should throw the same checked exception or its parent otherwise we will get compile time error.
- But there are no restrictions for un-checked exceptions.

28) Is it possible to overriding constructor?

- “Inheritance concept is not applicable for constructors and hence overriding concept also not applicable to the constructors. But constructors can be overloaded”.

- We can take constructor in any java class including abstract class also but **we can't take constructorinside inheritance**.

29) Is it possible to overriding non-abstract method as abstract?

- We should override Parent class abstract methods in Child classes to provide implementation.

Example:

```
abstract class Parent
{
        public abstract void methodOne();
}
class Child extends Parent
{
        public void methodOne()
        {}
}
```

- We can override Parent class non abstract method as abstract to stop availability of Parent class method implementation to the Child classes.

Example:

```
class Parent
{
public void methodOne()
{}
}
abstract class Child extends Parent
{
public abstract void methodOne();
}
```

30) Is it possible to override a var-arg method with a general method?

- A var-arg method should be overridden with var-arg method only. If we are trying to override with normal method then it will become overloading but not overriding.

Example:

```
class Parent
{
        public void methodOne(int... i)
        {
        System.out.println("parent class");
        }
}
class Child extends Parent
{
        public void methodOne(int i)
```

overloading but not overriding.

```
        {
        System.out.println("child class");
        }
}
class Test
{
        public static void main(String[] args)
        {
        Parent p=new Parent();
        p.methodOne(10);//parent class
        Child c=new Child();
        c.methodOne(10);//child class
        Parent p1=new Child();
        p1.methodOne(10);//parent class
        }
}
```

- In the above program if we replace child class method with var-arg then it will become overriding. In this case the output is

Parent class

Child class

Child class

31) Is it possible to override static method?

Overriding with respect to static methods:

Case 1:

- We can't override a static method as non static.

Example:

```
class Parent
{
        public static void methodOne()//here static methodOne() method is a class level
        {}
}
class Child extends Parent
{
        public void methodOne()//heremethodOne() method is a object level hence we can'toverride methodOne() method
        {}
}
```

Case 2:

- Similarly we can't override a non static method as static.

Case 3:

```
class Parent
```

```
{
public static void methodOne()
{}
}
class Child extends Parent
{
public static void methodOne()
{}
}
```

- It is valid. It seems to be overriding concept is applicable for static methods but it is not overriding it is method hiding.

32) What is method hiding?

- All rules of method hiding are exactly same as overriding except the following differences.

33) Explain differences between overriding and method hiding?

Overriding	Method hiding
1. Both Parent and Child class methods should be non static.	1. Both Parent and Child class methods should be static.
2. Method resolution is always takes care by JVM based on runtime object.	2. Method resolution is always takes care by compiler based on reference type.
3. Overriding is also considered as runtime polymorphism (or) dynamic polymorphism (or) late binding.	3. Method hiding is also considered as compile time polymorphism (or) static polymorphism (or) early biding.

34) Is it possible to override main() method?

- It seems to be overriding concept is applicable for static methods but it is not overriding it is method hiding.

35) Is it possible to override a variable?

- Overriding concept is not applicable for variables.
- Variable resolution is always takes care by compiler based on reference type.

Example:

```
class Parent
{
int x=888;
}
class Child extends Parent
{
int x=999;
}
class Test
```

```
{
                public static void main(String[] args)
                {
                        Parent p=new Parent();
                        System.out.println(p.x);//888
                        Child c=new Child();
                        System.out.println(c.x);//999
                        Parent p1=new Child();
                        System.out.println(p1.x);//888
                }
}
```

Note: In the above program Parent and Child class variables, whether both are static or non static whether one is static and the other one is non static there is no change in the answer.

36) Who is responsible for variable resolution?

- Variable resolution is always takes care by compiler based on reference type.

37) What are key differences between overriding and overloading?

38) Differences between overloading and overriding?

Property	Overloading	Overriding
1) Method names	1) Must be same.	1) Must be same.
2) Argument type	2) Must be different(at least order)	2) Must be same including order.
3) Method signature	3) Must be different.	3) Must be same.
4) Return types	4) No restrictions.	4) Must be same until 1.4v but from 1.5v onwards we can take co-variant return types also.
5) private,static,final methods	5) Can be overloaded.	5) Can not be overridden.
6) Access modifiers	6) No restrictions.	6) Weakering/reducing is not allowed.
7) Throws clause	7) No restrictions.	7) If child class method throws any checked exception compulsory parent class method should throw the same checked exceptions or its parent but no restrictions for un-checked exceptions.
8) Method resolution	8) Is always takes care by compiler based on referenced type.	8) Is always takes care by JVM based on runtime object.

9) Also known as	9) Compile time polymorphism (or) static(or)early binding.	9) Runtime polymorphism (or) dynamic (or) late binding.

39) Whenever we are loading child class is it required to load parent class?

- By default JVM loaded parent class explicitly no need.

40) Whenever we are loading parent class is it required to load child class?

- No need.

41) What is purpose of static block and explain its requirement with examples?

Static block:

- Static blocks will be executed at the time of class loading hence if we want to perform any activity at the time of class loading we have to define that activity inside static block.
- Every JDBC driver class internally contains a static block to register the driver with DriverManager hence programmer is not responsible to define this explicitly.

Example:

```
class Driver
{
        static
        {
                Register this driver with DriverManager
        }
```

42) Is it possible to take multiple static blocks in the same java class?

- With in a class we can take any no. Of static blocks and all these static blocks will be executed from top to bottom.

43) When the static blocks will be executed?

- Static blocks will be executed at the time of class loading

44) Without using main() method is it possible to print some statements to the console?

Ans: Yes, by using static block.

Example:

```
class Google
{
        static
        {
                System.out.println("hello i can print");
                System.exit(0);
        }
}
```

Output:

```
Hello i can print
```

45) Without using main() method and static blocks is it possible to print some statements to the console?

Example 1:

```
class Test
{
	static int i=methodOne();
	public static int methodOne()
	{
		System.out.println("hello i can print");
		System.exit(0);
		return 10;
	}
}
```

Output:

Hello i can print

Example 2:

```
class Test
{
	static Test t=new Test();
	Test()
	{
		System.out.println("hello i can print");
		System.exit(0);
	}
}
```

Output:

Hello i can print

Example 3:

```
class Test
{
	static Test t=new Test();
	{
	System.out.println("hello i can print");
	System.exit(0);
	}
}
```

Output:

Hello i can print

46) Without using System.out.println() statement is it possible to print some statement to the console?

Example:

```
class Test
```

```
{
	public static void main(String[] args)
	{
		System.err.println("hello");
	}
}
```

47) What is instance block when it will be executed?

- Instance block will be executed automatically for every object creation.
- The main objective ofInstance block is if we want to perform any activity for every object creation we have to define that activity inside instance block.

48) Difference between instance block and static block?

Instance block	static block
1) Instance block will be executed automatically for every object creation.	**1)** Static blocks will be executed at the time of class loading
2) The main objective of Instance block is if we want to perform any activity for every object creation we have to define that activity inside instance block.	2) The main objective of static block is if we want to perform any activity at the time of class loading we have to define that activity inside static block.

49) Difference between constructor and instance block?

Constructor	instance block
1) Constructor will be executed for every object creation.	**1)** Instance block will be executed for every object creation.
2) The main objective of constructor isto perform initialization of an object.	**2)** The main objective of Instance block is if we want to perform any activity for every object creation we have to define that activity inside instance block.
3) Constructor can take arguments.	**3)** Instance block can't take any arguments.

50) What is coupling?

Coupling:

- The degree of dependency between the components is called coupling.

Example:

```
class A
{
	static int i=B.j;
}
class B extends A
{
	static int j=C.methodOne();
```

```
}
class C extends B
{
        public static int methodOne()
        {
                return D.k;
        }
}
class D extends C
{
        static int k=10;
        public static void main(String[] args)
        {
                D d=new D();
        }
}
```

- The above components are said to be tightly coupled to each other because the dependency between the components is more.
- Tightly coupling is not a good programming practice because it has several serious disadvantages.

1) Without effecting remaining components we can't modify any component hence enhancement(development) will become difficult.
2) It reduces maintainability of the application.
3) It doesn't promote reusability of the code.

- It is always recommended to maintain loosely coupling between the components.

51) Is it recommended tight coupling (or) loose coupling?

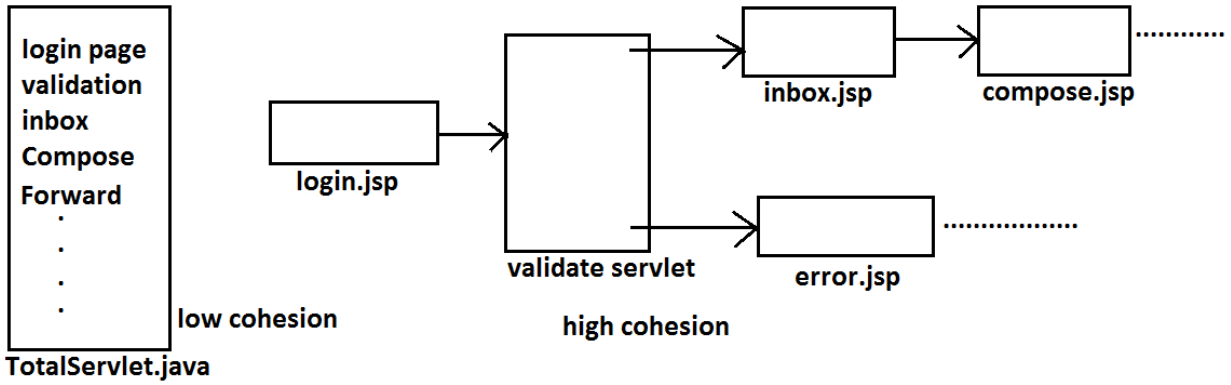
- Tightly coupling is not a good programming practice because it has several serious disadvantages.
- Without effecting remaining components we can't modify any component hence enhancement(development) will become difficult.
- It reduces maintainability of the application.
- It doesn't promote reusability of the code.
- It is always recommended to maintain loosely coupling between the components.

52) What is cohesion?

Cohesion:

- For every component we have to maintain a clear well defined functionality such type of component is said to be follow high cohesion.

Diagram:



53) Is it recommended high cohesion (or) low cohesion?

- High cohesion is always good programming practice because it has several advantages.
- Without effecting remaining components we can modify any component hence enhancement will become very easy.
- It improves maintainability of the application.
- It promotes reusability of the application.

Note: It is highly recommended to follow loosely coupling and high cohesion.

54) Explain about object type casting and what are various rules we have to follow while performing type casting?

Type casting:

- Parent class reference can be used to hold Child class object but by using that reference we can't call Child specific methods.

Example:

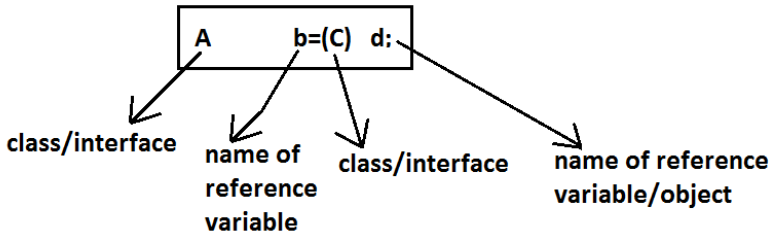
```
Object o=new String("bhaskar");//valid
System.out.println(o.hashCode());//valid
System.out.println(o.length());//C.E:cannot find symbol,symbol  : method length(),location: class java.lang.Object
```

- Similarly we can use interface reference to hold implemented class object.

Example:

Runnable r=new Thread();

Type casting syntax:



Compile time checking:

Rule 1: The type of "d" and "c" must have some relationship [either Child to Parent (or) Parent to Child (or) same type] otherwise we will get compile time error saying inconvertible types.

Example 1:

```
Object o=new String("bhaskar");
StringBuffer sb=(StringBuffer)o;
```
(valid)

Example 2:

```
String s=new String("bhaskar");
StringBuffer sb=(StringBuffer)s;
```
(inval id)

output:

```
compile time error
E:\scjp>javac Test.java
Test.java:6: inconvertible types
found   : java.lang.String
required: java.lang.StringBuffer
        StringBuffer sb=(StringBuffer)s;
```

Rule 2: "C" must be either same (or) derived type of "A" otherwise we will get compile time error saying incompatible types.

Found: C

Required: A

Example 1:

```
Object o=new String("bhaskar");
StringBuffer sb=(StringBuffer)o;
```
(valid)

Example 2:

```
Object o=new String("bhaskar");
StringBuffer sb=(String)o;
```
(invalid)

output:

```
compile time error
E:\scjp>javac Test.java
Test.java:6: incompatible types
found   : java.lang.String
required: java.lang.StringBuffer
        StringBuffer sb=(String)o;
```

Runtime checking:

- The underlying object type of "d" must be either same (or) derived type of "C" otherwise we will get runtime exception saying ClassCastException.

Example:

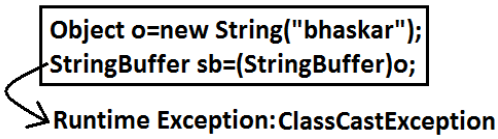
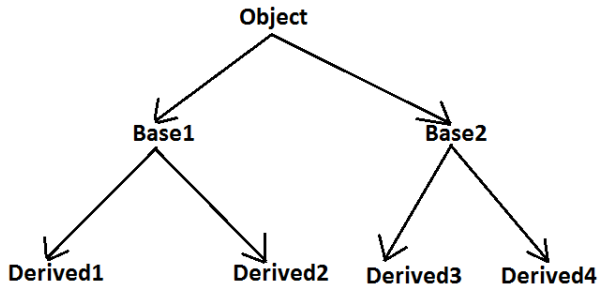


Diagram:



Base1 b=new Derived2();//valid

Object o=(Base1)b;//valid

Object o1=(Base2)o;//invalid

Object o2=(Base2)b;//invalid

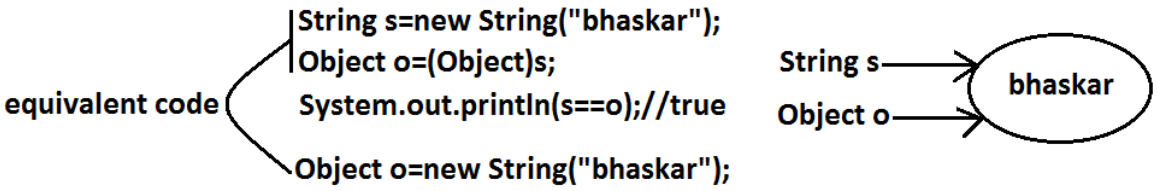
Base2 b1=(Base1)(new Derived1());//invalid

Base2 b2=(Base2)(new Derived3());//valid

Base2 b2=(Base2)(new Derived1());//invalid

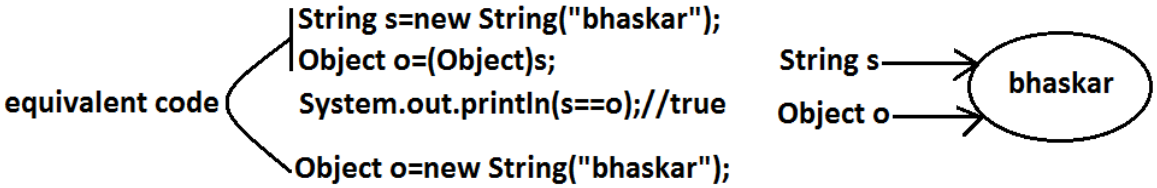
- Through Type Casting just we are converting the type of object but not object itself that is we are performing type casting but not object casting.

Example:



55) Is it possible to type cast parent object to the child type?

56) Is it possible to type cast child object to the parent type?



57) Explain the main important oops concepts?

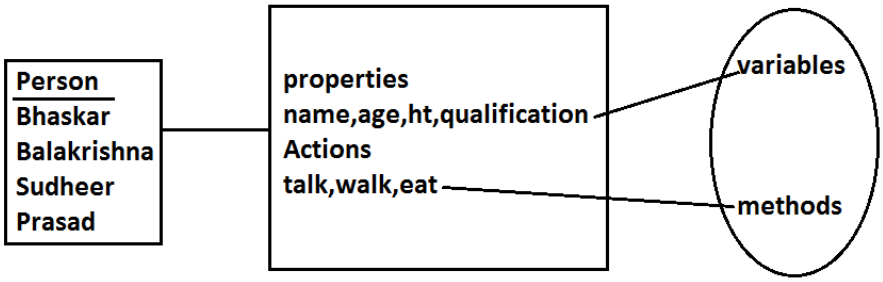
- The main important oops concepts (or) features are:
  - **Class/object**
  - **Abstraction**
  - **Encapsulation**
  - **Inheritance**

➢ **Polymorphism**

Class:

- A class is a group name that specifies properties and actions of object.

Diagram:

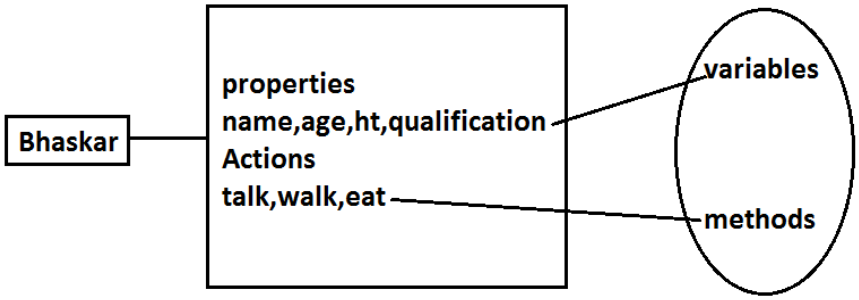


- Class also contains properties and actions. That means a class contains variables and methods.
- Objects exit physically but class does not exist physically.

Object:

- An object is anything that exists physically in the real world.
- An object contains properties and can perform actions.

Diagram:



- Properties are represented by variables.
- Actions are represented by methods.
- An object contains variables and methods.

Another definition:

- An object is an instance(physical form) of a class.
- A class is a plan (or) model for creating the objects.
- An object does not exist without a class but a class can exist without any object.

Data Hiding:

- Our internal data should not go out directly that is outside person can't access our internal data directly.
- By using private modifier we can implement data hiding.

Example:

```
class Account
```

```
{
        private double balance;
        .....;
        .....;
}
```

- The main advantage of data hiding is security.

Note: Recommended modifier for data members is private.

Abstraction:

- Hide internal implementation and just highlight the set of services, is called abstraction.
- By using abstract classes and interfaces we can implement abstraction.

Example:

- By using ATM GUI screen bank people are highlighting the set of services what they are offering without highlighting internal implementation.
- The main advantages of Abstraction are:

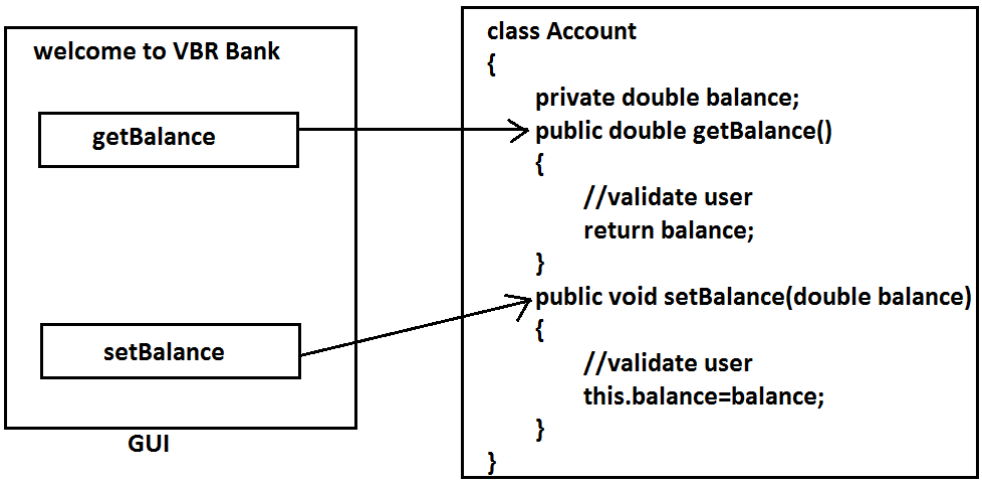
6) We can achieve security as we are not highlighting our internal implementation.
7) Enhancement will become very easy because without effecting end user we can able to perform any type of changes in our internal system.
8) It provides more flexibility to the end user to use system very easily.
9) It improves maintainability of the application.

Encapsulation:

- It is the process of Encapsulating data and corresponding methods into a single module.
- If any java class follows data hiding and abstraction such type of class is said to be encapsulated class.

Encapsulation=Datahiding+Abstraction

Example:



- In encapsulated class we have to maintain getter and setter methods for every data member.

- The main advantages of encapsulation are:

5) We can achieve security.
6) Enhancement will become very easy.
7) It improves maintainability of the application.
8) It provides flexibility to the user to use system very easily.

- The main disadvantage of encapsulation is it increases length of the code and slows down execution.

Inheritance:

- Producing new classes from existing classes is called inheritance.
- By using extends keyword we can implement inheritance.
- The main advantage of IS-A relationship is reusability.

Example:

```
class Parent
{
        public void methodOne()
        {}
}
class Child extends Parent
{
        public void methodTwo()
        {}
}
class Test
{
    public static void main(String[] args)
    {
        Parent p=new Parent();
        p.methodOne();
        p.methodTwo();
        Child c=new Child();
        c.methodOne();
        c.methodTwo();
        Parent p1=new Child();
        p1.methodOne();
        p1.methodTwo();
        Child c1=new Parent();
    }
}
```

C.E: cannot find symbol
symbol : method methodTwo()
location: class Parent

C.E: incompatible types
found : Parent
required: Child

Conclusion:

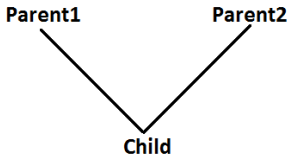
4) Whatever the parent has by default available to the child but whatever the child has by default not available to the parent. Hence on the child reference we can call both parent and child class methods. But on the parent reference we can call only methods available in the parent class and we can't call child specific methods.

5) Parent class reference can be used to hold child class object but by using that reference we can call only methods available in parent class and child specific methods we can't call.
6) Child class reference cannot be used to hold parent class object.

Multiple inheritances:

- Having more than one Parent class at the same level is called multiple inheritance.

Example:



- Any class can extends only one class at a time and can't extends more than one class simultaneously hence java won't provide support for multiple inheritance.

Example:

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

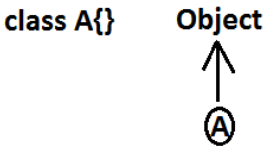
- But an interface can extends any no. Of interfaces at a time hence java provides support for multiple inheritances through interfaces.

Example:

```
interface A{}
interface B{}
interface C extends A,B{}
```

- If our class doesn't extends any other class then only our class is the direct child class of object.

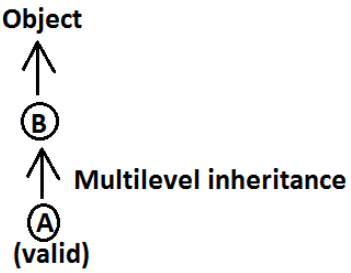
Example:



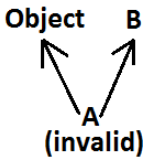
- If our class extends any other class then our class is not direct child class of object.

Example 1:

```
class B{}
class  A extends B{}
```



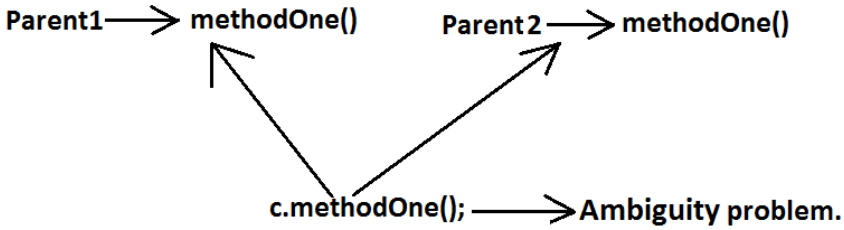
Example 2:



58) Why java won't provide support for multiple inheritances?

- There may be a chance of raising ambiguity problems.

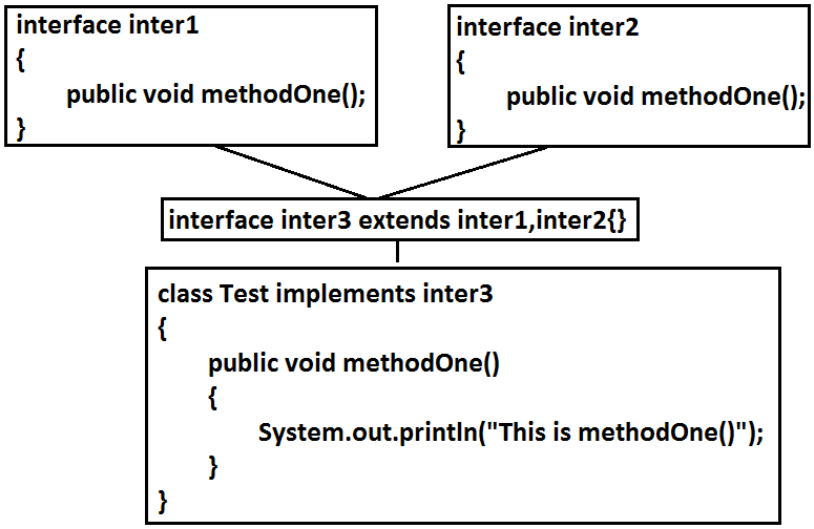
Example:



59) Why ambiguity problem won't be there in interfaces?

- Interfaces having dummy declarations and they won't have implementations hence no ambiguity problem.

Example:

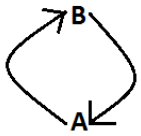


Cyclic inheritance:

- Cyclic inheritance is not allowed in java.

Example 1:

class A extends B{} } (invalid)
class B extends A{} } C.E:cyclic inheritance involving A



Example 2:

class A extends A{} —C.E→ cyclic inheritance involving A

Polymorphism:

- The ability to exist in various forms is called polymorphism.

60) What is the purpose of constructor?

- Constructor will be executed for every object creation.
- The main objective of constructor isto perform initialization of an object.

61) Explain about default constructor and its prototype?

Default constructor:

- For every class in java including abstract classes also constructor concept is applicable.
- If we are not writing at least one constructor then compiler will generate default constructor.
- If we are writing at least one constructor then compiler won't generate any default constructor. Hence every class contains either compiler generated constructor (or) programmer written constructor but not both simultaneously.

Prototype of default constructor:

1) It is always no argument constructor.
2) The access modifier of the default constructor is same as class modifier. (This rule is applicable only for public and default).
3) Default constructor contains only one line. **super();** it is a no argument call to super class constructor.

62) Is compiler generates default constructor always?

- If we are not writing at least one constructor then compiler will generate default constructor.
- If we are writing at least one constructor then compiler won't generate any default constructor.

63) Is it possible to take return type for the constructor?

- Return type concept is not applicable for constructor even void also by mistake if we are declaring the return type for the constructor we won't get any compile time error and runtime error compiler simply treats it as a method.

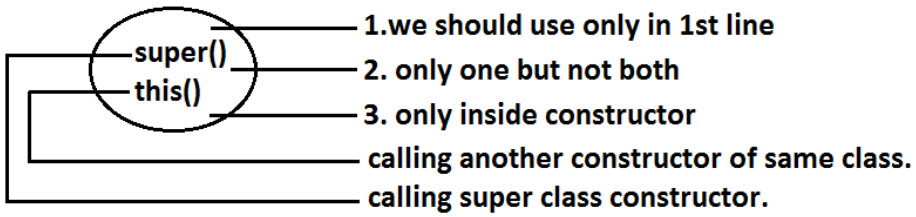
64) If we are taking return type to the constructor what will happen?

- If we are declaring the return type for the constructor we won't get any compile time error and runtime error compiler simply treats it as a method.

65) Explain about super() and this()?

- These are constructors calls.
- We should use only inside constructors.

Diagram:



66) Is it possible to overload constructors?

Overloaded constructors:

- A class can contain more than one constructor and all these constructors having the same name but different arguments and hence these constructors are considered as overloaded constructors.

Example:

```
class Test
{
	Test(double d)
	{
		this(10);
		System.out.println("double-argument constructor");
	}
	Test(int i)
	{
		this();
		System.out.println("int-argument constructor");
	}
	Test()
	{
		System.out.println("no-argument constructor");
	}
	public static void main(String[] args)
	{
		Test t1=new Test(10.5);//no-argument constructor/int-argument
constructor/double-argument constructor
		Test t2=new Test(10);//no-argument constructor/int-argument constructor
		Test t3=new Test();//no-argument constructor
	}}
```

67) Is inheritance concept applicable for constructor?

- Inheritance concept is not applicable for constructors and hence overriding concept also not applicable to the constructors.

68) What are various modifiers applicable for constructors?

- The only applicable modifiers for the constructors are **public, default, private, protected**.
- If we are using any other modifier we will get compile time error.

69) Where we can use private constructors?

- To create singleton classes we have to use private constructor.

70) What is singleton class and give an example?

Singleton classes:

- For any java class if we are allow to create only one object such type of class is said to be singleton class.

Example:

1) Runtime class
2) ActionServlet
3) ServiceLocator

Creation of our own singleton classes:

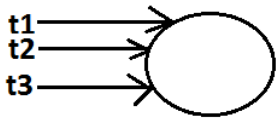
- We can create our own singleton classes for this we have to use private constructor and factory method.

Example:

```
class Test
{
	private static Test t=null;
	private Test()
	{}
	public static Test getTest()//getTest() method is a factory method
	{
		if(t==null)
		{
			t=new Test();
		}
		return t;
	}
}
class Client
{
	public static void main(String[] args)
```

```
	{
		System.out.println(Test.getTest().hashCode());//1671711
		System.out.println(Test.getTest().hashCode());//1671711
		System.out.println(Test.getTest().hashCode());//1671711
		System.out.println(Test.getTest().hashCode());//1671711
	}
}
```

Diagram:



71) Is it possible to create our own singleton class and explain the process?

- We can create any xxxton classes like(double ton,trible ton….etc).

Example:

```
class Test
{
	private static Test t1=null;
	private static Test t2=null;
	private Test()
	{}
	public static Test getTest()//getTest() method is a factory method
	{
		if(t1==null)
		{
			t1=new Test();
			return t1;
		}
		else if(t2==null)
		{
			t2=new Test();
			return t2;
		}
		else
		{
			if(Math.random()<0.5)
				return t1;
			else
				return t2;
		}
	}
}
class Client
{
```

```
        public static void main(String[] args)
        {
                System.out.println(Test.getTest().hashCode());//1671711
                System.out.println(Test.getTest().hashCode());//11394033
                System.out.println(Test.getTest().hashCode());//11394033
                System.out.println(Test.getTest().hashCode());//1671711
        }
}
```

72) What is factory method?

Factory method:

- By using class name if we are calling a method and that method returns the same class object such type of method is called factory method.

Example:

```
Runtime r=Runtime.getRuntime();//getRuntime is a factory method.
DateFormat df=DateFormat.getInstance();
```

- If object creation required under some constraints then we can implement by using factory method.

73) If parent class constructor throws some checked exception is it compulsory to require to throw that exception by child class constructor?

- If Parent class constructor throws some checked exception compulsory Child class constructor should throw the same checked exception (or) its Parent.

Example:

```
class Parent
{
        Parent()throws java.io.IOException
        {}
}
class Child extends Parent
{
        Child()throws Exception
        {
                super();
        }
}
```

74) Is the first line in constructor is always super?

- We can use either super() (or) this() but not both simultaneously.

75) Is it possible to write a constructor in abstract class?

- Yes, it is possible there is no restriction.

76) We can't create an object of abstract class what is the need of constructor in abstract class?

- Abstract class constructor will be executed to perform initialization of child class object.

77) Is it possible to place constructor in an interface?

- No it is not possible to place constructor in an interface.

Exception Handling

1) What **is an exception?**

Exception: An unwanted unexpected event that disturbs normal flow of the program is called exception.

Example:

SleepingException
TyrePunchuredException
FileNotFoundException…..etc

- It is highly recommended to handle exceptions.

2) What is the purpose of exception handling?

- The main objective of exception handling is graceful (normal) termination of the program.

3) What is the meaning of exception handling?

- Exception handling doesn't mean repairing an exception. We have to define alternative way to continue rest of the program normally this way of "defining alternative is nothing but exception handling".

Example: Suppose our programming requirement is to read data from London file at runtime if London file is not available our program should not be terminated abnormally. We have to provide a local file to continue rest of the program normally. This way of defining alternative is nothing but exception handling.

Program:

```
Try
{
read data from london file
}
catch(FileNotFoundException e)
{
use local file and continue rest of the program normally
}}
```

4) Explain default exception handling mechanism in java?

Default exception handling in java:

1) If an exception raised inside any method then the method is responsible to create Exception object with the following information.
    1) Name of the exception.
    2) Description of the exception.
    3) Location of the exception.
2) After creating that Exception object the method handovers that object to the JVM.
3) JVM checks whether the method contains any exception handling code or not. If method won't contain any handling code then JVM terminates that method abnormally and removes corresponding entry form the stack.
4) JVM identifies the caller method and checks whether the caller method contain any handling code or not. If the caller method also does not contain handling code then JVM terminates that caller also abnormally and the removes corresponding entry from the stack.
5) This process will be continued until main() method and if the main() method also doesn't contain any exception handling code then JVM terminates main() method and removes corresponding entry from the stack.
6) Then JVM handovers the responsibility of exception handling to the default exception handler.
7) Default exception handler just print exception information to the console in the following formats and terminates the program abnormally.

    Name of exception: description
    Location of exception (stack trace)

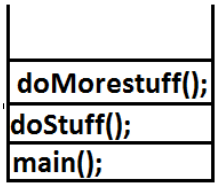
Example:

```
class Test{
public static void main(String[] args){
doStuff();
}
public static void doStuff(){
doMoreStuff();
}
public static void doMoreStuff(){
System.out.println(10/0);
}}
```

Output:

```
Runtime error
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at Test.doMoreStuff(Test.java:10)
      at Test.doStuff(Test.java:7)
      at Test.main(Test.java:4)
```

Diagram:



5) What is the purpose of try?

Try: To maintain risky code.

6) What is the purpose of catch block?

Catch: To maintain handling code.

7) Is try with multiple catch block is possible?

Try with multiple catch blocks: The way of handling an exception is varied from exception to exception hence for every exception raise a separate catch block is required that is try with multiple catch blocks is possible and recommended to use.

Example:

Try
{
.
.
.
.
}
catch(Exception e)
	try
	{
	.
	.
	.
	.
	catch(FileNotFoundException e)
	{

{
default handler
}
 	
● This approach is not recommended because for any type of Exception we are using the same catch block.
	use local file
	}
	catch(ArithmeticException e)
	{
	perform these Arithmetic operations
	}
	catch(SQLException e)
	{
	don't use oracle db, use mysql db
	}
	catch(Exception e)
	{
	default handler
	}
	
	● This approach is highly recommended because for any exception raise we are defining a separate catch block.

8) If try with multiple catch block present is order of catch blocks impotent in which order we have to take?

- If try with multiple catch blocks presents then order of catch blocks is very important it should be from child to parent by mistake if we are taking from parent to child then we will get Compile time error saying "exception xxx has already been caught".

Example:

class Test{
public static void main(String[] args)
{
try
{
System.out.println(10/0);
}
catch(Exception e)
{
e.printStackTrace();
}
	class Test
	{
	public static void main(String[] args)
	{
	try
	{
	System.out.println(10/0);
	}
	catch(ArithmeticException e)
	{
	e.printStackTrace();

catch(ArithmeticException e)
{
e.printStackTrace();
}}}
Output:
Compile time error.
Test.java:13: exception
java.lang.ArithmeticException has already been caught
catch(ArithmeticException e)
	}
	catch(Exception e)
	{
	e.printStackTrace();
	}}}
	Output:
	Compile successfully.

9) What are various methods to print exception information? And differentiate them?

Various methods to print exception information:

- Throwable class defines the following methods to print exception information to the console.

printStackTrace(): This method prints exception information in the following format.

Name of the exception: description of exception Stack trace

toString(): This method prints exception information in the following format.

Name of the exception: description of exception

getMessage(): This method returns only description of the exception.

Description.

Example:

```
class Test
{
public static void main(String[] args){
try
{
System.out.println(10/0);
}
catch(ArithmeticException e)
{
e.printStackTrace();
System.out.println(e);
System.out.println(e.getMessage());
}}}
```

```
java.lang.ArithmeticException: / by zero
        at Test.main(Test.java:6)
java.lang.ArithmeticException: / by zero
/ by zero
```

Note: Default exception handler internally uses printStackTrace() method to print exception information to the console.

10) If an exception raised inside catch block then what will happen?

- If an exception raised inside catch block then it's always abnormal termination but before the finally block will be executed.

11) Is it possible to take try, catch inside try block?

Example:

```
class Test{
public static void main(String[] args){
try
{  try{}
   catch(Exception e){}
}
catch(Exception e)
{}
}}
```

Output:

Compile and running successfully.

12) Is it possible to take try, catch inside catch block?

Example:

```
class Test{
public static void main(String[] args){
try
{ }
catch(Exception e1)
{
 try{}
 catch(Exception e2){}
}}}
```

13) Is it possible to take try without catch?

Example:

```
class Test{
public static void main(String[] args){
try
{}
}}
```

Output:

```
Compile time error
Test1.java:3: 'try' without 'catch' or 'finally'
Try
```

14) What is the purpose of finally block?

Finally block:

- It is never recommended to take clean up code inside try block because there is no guarantee for the execution of every statement inside a try.
- It is never recommended to place clean up code inside catch block because if there is no exception then catch block won't be executed.
- We require some place to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and whether handled or not handled such type of place is nothing but finally block.
- Hence the main objective of finally block is to maintain cleanup code.

15) Is finally block will be execute always?

- The specialty of finally block is it will be executed always irrespective of whether the exception raised or not raised and whether handled or not handled.

16) In which situation finally block will not executed?

- There is only one situation where the finally block won't be executed is whenever we are using System.exit(0) method.

17) If return statement present inside try is finally block will be executed?

- Even though return statement present in try or catch blocks first finally block will be executed and after that only return statement will be considered that is finally block dominates return statement.

Example:

```
class Test
{
public static void main(String[] args)
{
try
{
System.out.println("try block executed");
return;
}
catch(ArithmeticException e)
{
System.out.println("catch block executed");
}
finally
{
System.out.println("finally block executed");
}}}
```

Output:

```
Try block executed
Finally block executed
```

18) Is it possible to write any statement between try-catch and finally?

Example:

```
class Test{
public static void main(String[] args){
try
{}
catch(Exception e)
{}
System.out.println("hello");
finally
{}
}}+
```

Output:

```
Compile time error.
Test1.java:8: 'finally' without 'try'
Finally
```

19) Is it possible to take two finally blocks for the same try?

Example:

```
class Test{
public static void main(String[] args){
try
{}
finally
{}
finally
{}
}}
```

Output:

```
Compile time error.
Test1.java:7: 'finally' without 'try'
Finally
```

20) Is syntax try-finally-catch is valid?

Example:

```
class Test1{
public static void main(String[] args){
try
```

```
9{}
finally
{}
catch(Exception e)
{}
}}
```

Output:

```
Compile time error.
Test1.java:7: 'catch' without 'try'
catch(Exception e)
```

21) What is the purpose of throw?

- In general we can use throw keyword for customized exceptions but not for predefined exceptions.

22) Is it possible to throw an error?

- Most of the cases errors are not caused by our program these are due to lack of system resources and these are non recoverable.

23) Is it possible to throw any java object?

- We can use throw keyword only for Throwable types otherwise we will get compile time error saying incomputable types.

24) After throw is it allow to take any statement directly?

- After throw statement we can't take any statement directly otherwise we will get compile time error saying unreachable statement.

Example:

```
class Test3
{
public static void main(String[] args){
System.out.println(10/0);
System.out.println("hello");
}}
```

Output:

Runtime error: Exception in thread "main" java.lang.ArithmeticException: / by zero
at Test3.main(Test3.java:4)

```
class Test3
{
public static void main(String[] args){
throw new ArithmeticException("/ by zero");
System.out.println("hello");
}}
```

Output:

```
Compile time error.
Test3.java:5: unreachable statement
System.out.println("hello");
```

25) What is the purpose of throws?

- The main objective of "throws" keyword is to delicate(In order to avoid trouble) the responsibility of exception handling to the caller method.
- "Throws" keyword required only for checked exceptions. Usage of throws for unchecked exception there is no use.

26) What is the difference between throw and throws?

Throw:

- In general we can use throw keyword for customized exceptions but not for predefined exceptions.

Throws:

- The main objective of "throws" keyword is to delicate the responsibility of exception handling to the caller method.
- "Throws" keyword required only checked exceptions. Usage of throws for unchecked exception there is no use.

27) What is the difference between throw and thrown?

Throw:

- In general we can use throw keyword for customized exceptions but not for predefined exceptions.

Thrown:

- Thrown keyword is not available in java.

28) Is it possible to use throws keyword for any java class?

- We can use throws keyword only for Throwable types otherwise we will get compile time error saying incompatible types.

Example:

class Test3{
public static void main(String[] args)throws
Test3
{}
}
Output:
Compile time error
Test3.java:2: incompatible types
found : Test3
required: java.lang.Throwable
public static void main(String[] args)throws
Test3
	class Test3 extends RuntimeException{
	public static void main(String[] args)throws
	Test3
	{}
	}
	Output:
	Compile and running successfully.

29) If we are taking catch block for an exception but there is no chance of raising that exception in try then what will happen?

- Application will terminate normally and there are no compile and runtime errors**.**

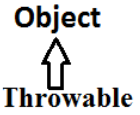
30) Explain exception handling keywords?

1) **Try:** To maintain risky code.
2) **Catch:** To maintain handling code.
3) **Finally:** To maintain cleanup code.

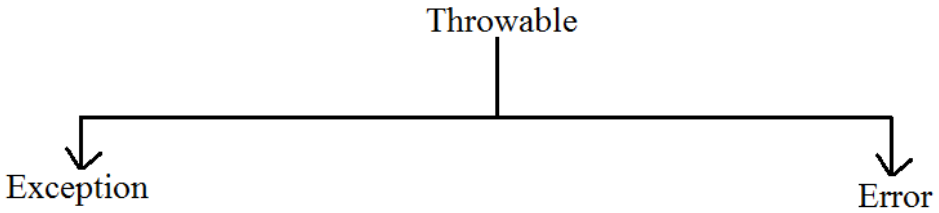
4) **Throw:** To handover our created exception object to the JVM manually.
5) **Throws:** To delegate responsibility of exception handling to the caller method.

31) Which class acts as root for entire java exception hierarchy?

Exception hierarchy:



- Throwable acts as a root for exception hierarchy.
- Throwable class contains the following two child classes.



32) What is the difference between error and exception?

Exception: Most of the cases exceptions are caused by our program and these are recoverable.

Error: Most of the cases errors are not caused by our program these are due to lack of system resources and these are non recoverable.

33) What is the difference between checked exception and un-checked exception?

Checked Vs Unchecked Exceptions:

- The exceptions which are checked by the compiler for smooth execution of the program at runtime are called checked exceptions.
    1) HallTicketMissingException
    2) PenNotWorkingException
    3) FileNotFoundException
- The exceptions which are not checked by the compiler are called unchecked exceptions.
    1) BombBlaustException
    2) ArithmeticException
    3) NullPointerException

Note: RuntimeExceptionand its child classes, Error and its child classes are unchecked and all the remaining are considered as checked exceptions.

Note: Whether exception is checked or unchecked compulsory it should occur at runtime only there is no chance of occurring any exception at compile time.

34) What is the difference between partially checked and fully checked exception?

Partially checked Vs fully checked:

- A checked exception is said to be fully checked if and only if all its child classes are also checked.

Example:

1) IOException
2) InterruptedException

- A checked exception is said to be partially checked if and only if some of its child classes are unchecked.

Example: Exception

- The only partially checked exceptions available in java are:
    1. Throwable.
    2. Exception.

35) What is a customized exception?

Customized Exceptions (User defined Exceptions):

- Sometimes we can create our own exception to meet our programming requirements. Such type of exceptions are called customized exceptions (user defined exceptions).

Example:

1) InSufficientFundsException
2) TooYoungException
3) TooOldException

36) Explain the process of creating the customized exception?

Program:

```
class TooYoungException extends RuntimeException
{
TooYoungException(String s)
{
super(s);
}
}
class TooOldException extends RuntimeException
{
TooOldException(String s)
{
super(s);
}
}
class CustomizedExceptionDemo
{
public static void main(String[] args){
int age=Integer.parseInt(args[0]);
if(age>60)
{
```

```
throw new TooYoungException("please wait some more time.... u will get best match");
}
else if(age<18)
{
throw new TooOldException("u r age already crossed....no chance of getting married");
}
else
{
System.out.println("you will get match details soon by e-mail");
}}}
```

Output:

1) E:\scjp>java CustomizedExceptionDemo 61
   Exception in thread "main" TooYoungException: please wait some more time.... u will get best match
   at CustomizedExceptionDemo.main(CustomizedExceptionDemo.java:21)
2) E:\scjp>java CustomizedExceptionDemo 27
   You will get match details soon by e-mail
3) E:\scjp>java CustomizedExceptionDemo 9
   Exception in thread "main" TooOldException: u r age already crossed....no chance of getting married
   at CustomizedExceptionDemo.main(CustomizedExceptionDemo.java:25)

Note: It is highly recommended to maintain our customized exceptions as unchecked by extending RuntimeException.

37) Explain control flow in try, catch and finally?

38) Can you give the most common occurred exception in your previous project?

Top-10 Exceptions:

- Exceptions are divided into two types. They are:
  1) JVM Exceptions:
  2) Programatic exceptions:

JVM Exceptions:

- The exceptions which are raised automatically by the jvm whenever a particular event occurs.

Example:

1) ArrayIndexOutOfBoundsException(AIOOBE)
2) NullPointerException (NPE).

Programatic Exceptions:

- The exceptions which are raised explicitly by the programmer (or) by the API developer are called programatic exceptions.

Example:

1) IllegalArgumentException(IAE).

1) ArrayIndexOutOfBoundsException:

- It is the child class of RuntimeException and hence it is unchecked. Raised automatically by the JVM whenever we are trying to access array element with out of range index.

Example:

```
class Test{
public static void main(String[] args){
int[] x=new int[10];
System.out.println(x[0]);//valid
System.out.println(x[100]);//AIOOBE
System.out.println(x[-100]);//AIOOBE
}}
```

2) NullPointerException:

- It is the child class of RuntimeException and hence it is unchecked. Raised automatically by the JVM, whenever we are trying to call any method on null.

Example:

```
class Test{
public static void main(String[] args){
String s=null;
System.out.println(s.length());→R.E: NullPointerException
}}
```

3) StackOverFlowError:

- It is the child class of Error and hence it is unchecked. Whenever we are trying to invoke recursive method call JVM will raise StackOverFloeError automatically.

Example:

```
class Test
{
public static void methodOne()
{
methodTwo();
}
public static void methodTwo()
{
methodOne();
}
public static void main(String[] args)
{
```

```
methodOne();
}
}
```

Output:

Run time error: StackOverFloeError

4) **NoClassDefFound:**

- It is the child class of Error and hence it is unchecked. JVM will raise this error automatically whenever it is unable to find required .class file.

Example:java Test

- If Test.class is not available. Then we will get NoClassDefFound error.

5) **ClassCastException:**

- It is the child class of RuntimeException and hence it is unchecked. Raised automatically by the JVM whenever we are trying to typecast parent object to child type.

Example:

```
class Test
{
public static void main(String[] args)
{
String s=new String("bhaskar");
Object o=(Object)s;
}
}
```

output:
valid

```
class Test
{
public static void main(String[] args)
{
Object o=new Object();
String s=(String)o;
}
}
```

output:
Runtime exception:ClassCastException

```
class Test
{
public static void main(String[] args)
{
Object o=new String("bhaskar");
String s=(String)o;
}
}
```

output:
valid

6) **ExceptionInInitializerError:**

- It is the child class of Error and it is unchecked. Raised automatically by the JVM, if any exception occurs while performing static variable initialization and static block execution.

Example 1:

```
class Test{
static int i=10/0;
}
```

Output:

Runtime exception:

- Exception in thread "main" java.lang.ExceptionInInitializerError

Example 2:

```
class Test{
static {
String s=null;
System.out.println(s.length());
```

```
}}
```

Output:

Runtime exception:Exception in thread "main" java.lang.ExceptionInInitializerError

7) IllegalArgumentException:

- It is the child class of RuntimeException and hence it is unchecked. Raised explicitly by the programmer (or) by the API developer to indicate that a method has been invoked with inappropriate argument.

Example:

```
class Test{
public static void main(String[] args){
Thread t=new Thread();
t.setPriority(10);→valid
t.setPriority(100);→invalid
}}
```

Output:

Runtime exception

- Exception in thread "main" java.lang.IllegalArgumentException.

8) NumberFormatException:

- It is the child class of IllegalArgumentException and hence is unchecked. Raised explicitly by the programmer or by the API developer to indicate that we are attempting to convert string to the number. But the string is not properly formatted.

Example:

```
class Test{
public static void main(String[] args){
int i=Integer.parseInt("10");
int j=Integer.parseInt("ten");
}}
```

Output:

Runtime Exception

- Exception in thread "main" java.lang.NumberFormatException: For input string: "ten"

9) **IllegalStateException:**

- It is the child class of RuntimeException and hence it is unchecked. Raised explicitly by the programmer or by the API developer to indicate that a method has been invoked at inappropriate time.

Example:

- Once session expires we can't call any method on the session object otherwise we will get IllegalStateException

HttpSession session=req.getSession();
System.out.println(session.getId());
session.invalidate();
System.out.println(session.getId());→illgalstateException

10) **AssertionError:**

- It is the child class of Error and hence it is unchecked. Raised explicitly by the programmer or by API developer to indicate that Assert statement fails.

Example:

assert(false);

Exception/Error	Raised by
1) AIOOBE 2) NPE(NullPointerException) 3) StackOverFlowError 4) NoClassDefFoundError 5) CCE(ClassCastException) 6) ExceptionInInitializerError	Raised automatically by JVM(JVM Exceptions)
7) IAE(IllegalArgumentException) 8) NFE(NumberFormatException) 9) ISE(IllegalStateException) 10) AE(AssertionError)	Raised explicitly either by programmer or by API developer (Programatic Exceptions).

39) Explain the cases where you used exception handling in your previous project?

Multi Threading

1) What is multitasking?

Multitasking: Executing several tasks simultaneously is the concept of multitasking. There are two types of multitasking's.

1) **Process based multitasking.**
2) **Thread based multitasking.**

Diagram:

2) What is multithreading and explain its application areas?

Process based multitasking: Executing several tasks simultaneously where each task is a separate independent process such type of multitasking is called process based multitasking.

Example:

- While typing a java program in the editor we can able to listen mp3 audio songs at the same time we can download a file from the net all these tasks are independent of each other and executing simultaneously and hence it is Process based multitasking.
- This type of multitasking is best suitable at "**oslevel**".

Thread based multitasking: Executing several tasks simultaneously where each task is a separate independent part of the same program, is called Thread based multitasking. And each independent part is called a "Thread".

- This type of multitasking is best suitable for "programatic level".
- When compared with "C++", developing multithreading examples is very easy in java because java provides in built support for multithreading through a rich API (Thread, Runnable, ThreadGroup, ThreadLocal….etc).
- In multithreading on 10% of the work the programmer is required to do and 90% of the work will be down by java API.
- The main important application areas of multithreading are:
    1) To implement multimedia graphics.
    2) To develop animations.
    3) To develop video games etc.
- Whether it is process based or Thread based the main objective of multitasking is to improve performance of the system by reducing response time.

3) What is advantage of multithreading?

- The main objective of multitasking is to improve performance of the system by reducing response time.

4) When compared with C++ what is the advantage in java with respect to multithreading?

- When compared with "C++", developing multithreading examples is very easy in java because java provides in built support for multithreading through a rich API (Thread, Runnable, ThreadGroup, ThreadLocal….etc).

5) In how many ways we can define a thread?

- We can define a Thread in the following 2 ways.
    1. By extending Thread class.
    2. By implementing Runnable interface.

6) Among extending thread and implementing runnable which approach is recommend?

- Among the 2 ways of defining a Thread, implements Runnable approach is always recommended.
- In the 1st approach our class should always extends Thread class there is no chance of extending any other class hence we are missing the benefits of inheritance.

- But in the 2nd approach while implementing Runnable interface we can extend some other class also. Hence implements Runnable mechanism is recommended to define a Thread.

7) Difference between t.start() and t.run()?

- In the case of t.start() a new Thread will be created which is responsible for the execution of run() method. But in the case of t.run() no new Thread will be created and run() method will be executed just like a normal method by the main Thread. In the above program if we are replacing t.start() with t.run() the following is the output.

8) Explain about thread scheduler?

- If multiple Threads are waiting to execute then which Thread will execute 1st is decided by "Thread Scheduler" which is part of JVM.
- Which algorithm or behavior followed by Thread Scheduler we can't expect exactly it is the JVM vendor dependent hence in multithreading examples we can't expect exact execution order and exact output.

9) If we are not overriding run() what will happen?

- If we are not overriding run() method then Thread class run() method will be executed which has empty implementation and hence we won't get any output.
- It is highly recommended to override run() method. Otherwise don't go for multithreading concept.

10) Is it possible overloading of run()?

- We can overload run() method but Thread class start() method always invokes no argument run() method the other overload run() methods we have to call explicitly then only it will be executed just like normal method.

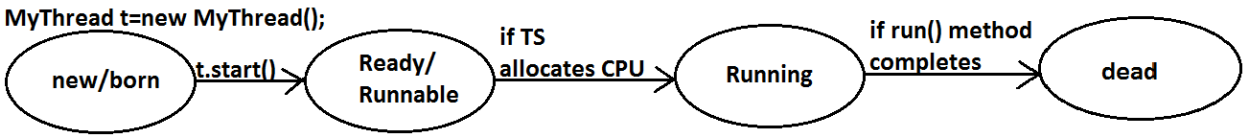
11) Is it possible to override a start() method and what will happen?

- If we override start() method then our start() method will be executed just like a normal method call and no new Thread will be started.

12) Explain life cycle of a thread?

Life cycle of the Thread:

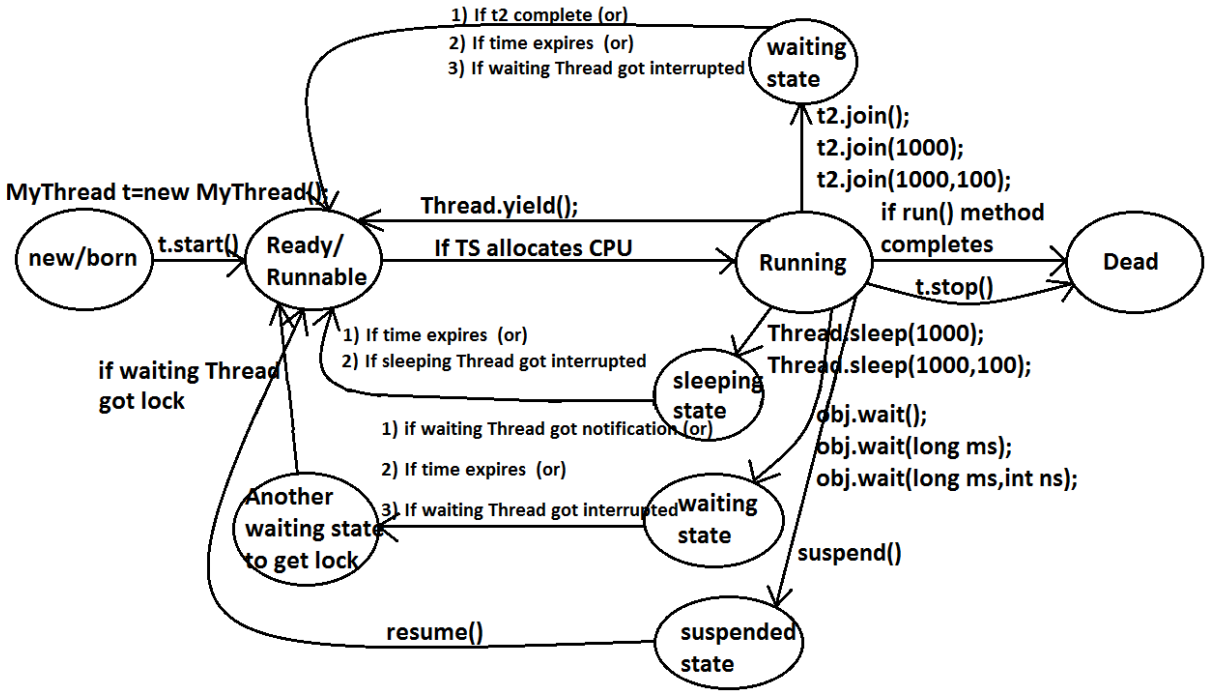
Diagram:



- Once we created a Thread object then the Thread is said to be in new state or born state.
- Once we call start() method then the Thread will be entered into Ready or Runnable state.

- If Thread Scheduler allocates CPU then the Thread will be entered into running state.
- Once run() method completes then the Thread will entered into dead state.

(Or)



13) What is the importance of thread class start() method?

- For every Thread the required mandatory activities like registering the Thread with Thread Scheduler will takes care by Thread class start() method and programmer is responsible just to define the job of the Thread inside run() method. That is start() method acts as best assistant to the programmer.
- We can conclude that without executing Thread class start() method there is no chance of starting a new Thread in java.

14) After starting a thread if we try to restart the same thread once again what will happen?

- After starting a Thread we are not allowed to restart the same Thread once again otherwise we will get runtime exception saying "IllegalThreadStateException".

15) Explain thread class constructors?

1) Thread t=new Thread();
2) Thread t=new Thread(Runnable r);
3) Thread t=new Thread(String name);
4) Thread t=new Thread(Runnable r,String name);
5) Thread t=new Thread(ThreadGroup g,String name);
6) Thread t=new Thread(ThreadGroup g,Runnable r);
7) Thread t=new Thread(ThreadGroup g,Runnable r,String name);

8) Thread t=new Thread(ThreadGroup g,Runnable r,String name,long stackSize);

16) How to get and set name of a thread?

Getting and setting name of a Thread:

- Every Thread in java has some name it may be provided explicitly by the programmer or automatically generated by JVM.
- Thread class defines the following methods to get and set name of a Thread.

Methods:

1) **public final String getName()**
2) **public final void setName(String name)**

Example:

```
class MyThread extends Thread
{}
class ThreadDemo
{
        public static void main(String[] args)
        {
                System.out.println(Thread.currentThread().getName());//main
                MyThread t=new MyThread();
                System.out.println(t.getName());//Thread-0
                Thread.currentThread().setName("Bhaskar Thread");
                System.out.println(Thread.currentThread().getName());//Bhaskar Thread
        }}
```

Note: We can get current executing Thread object reference by using Thread.currentThread() method.

17) Who uses thread priorities?

- Thread scheduler uses these priorities while allocating CPU.

18) Default priority for main thread?

- The default priority only for the main Thread is 5. But for all the remaining Threads the default priority will be inheriting from parent to child. That is whatever the priority parent has by default the same priority will be for the child also.

19) Once we create a new thread what is its priority?

- Whatever the priority parent has by default the same priority will be for the child also.

20) How to get priority from thread and set priority to a thread?

- We can get and set the priority of a Thread by using the following methods.

1) public final int getPriority()

2) public final void setPriority(int newPriority);//the allowed values are 1 to 10

21) If we are trying to set priority of thread as 100, what will happen?

- The allowed values are 1 to 10 otherwise we will get runtime exception saying "IllegalArgumentException".

22) If two threads having different priority then which thread will get chance first for execution?

- The Thread which is having highest priority will get chance for 1st execution.

23) If two threads having same priority then which thread will get chance first for execution?

- If 2 Threads having the same priority then we can't expect exact execution order it depends on Thread scheduler whose behavior is vendor dependent.

24) How we can prevent thread from execution?

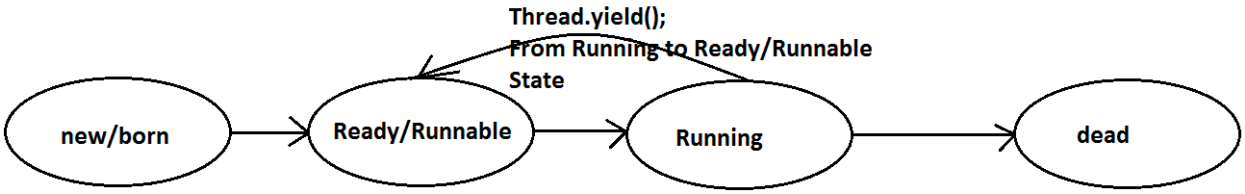
- We can prevent(stop) a Thread execution by using the following methods.
    1) **yield();**
    2) **join();**
    3) **sleep();**

25) What is yield() and explain its purpose?

yield():

- yield()(give up) method causes "to pause current executing Thread for giving the chance of remaining waiting Threads of same priority".
- If all waiting Threads have the low priority or if there is no waiting Threads then the same Thread will be continued its execution.
- If several waiting Threads with same priority available then we can't expect exact which Thread will get chance for execution.
- The Thread which is yielded when it get chance once again for execution is depends on mercy of the Thread scheduler.
- public static native void **yield**();

Diagram:



Example:

```
class MyThread extends Thread
{
        public void run()
        {
                for(int i=0;i<5;i++)
                {
```

```
                    Thread.yield();
                    System.out.println("child thread");
                }
        }
}
class ThreadYieldDemo
{
        public static void main(String[] args)
        {
                MyThread t=new MyThread();
                t.start();
                for(int i=0;i<5;i++)
                {
                        System.out.println("main thread");
                }
        }
}
```

Output:

main thread….4child thread….4

- In the above example the chance of completing main Thread 1st is high because child Thread always calling yield() method.

26) Is join is overloaded?

Join():

- If a Thread wants to wait until completing some other Thread then we should go for join() method.
- Yes, join() method is overloaded method. The following methods are overloaded methods.
    1) **public final void join()throws InterruptedException**
    2) **public final void join(long ms) throws InterruptedException**
    3) **public final void join(long ms,int ns) throws InterruptedException**

27) Purpose of sleep() method?

Sleep() method:

- If a Thread don't want to perform any operation for a particular amount of time then we should go for sleep() method.
    1) **public static native void sleep(long ms) throws InterruptedException**
    2) **public static void sleep(long ms,int ns)throws InterruptedException**

28) What is synchronized keyword? Explain its advantages and disadvantages?

- Synchronized is the keyword applicable for methods and blocks but not for classes and variables.
- If a method or block declared as the synchronized then at a time only one Thread is allow to execute that method or block on the given object.
- The main advantage of synchronized keyword is we can resolve date inconsistency problems.
- But the main disadvantage of synchronized keyword is it increases waiting time of the Thread and effects performance of the system.
- Hence if there is no specific requirement then never recommended to use synchronized keyword.

29) What is object lock and when it is required?

- Internally synchronization concept is implemented by using lock concept.
- Every object in java has a unique lock. Whenever we are using synchronized keyword then only lock concept will come into the picture.
- If a Thread wants to execute any synchronized method on the given object 1st it has to get the lock of that object. Once a Thread got the lock of that object then it's allow to execute any synchronized method on that object. If the synchronized method execution completes then automatically Thread releases lock.
- While a Thread executing any synchronized method the remaining Threads are not allowed execute any synchronized method on that object simultaneously. But remaining Threads are allowed to execute any non-synchronized method simultaneously. [Lock concept is implemented based on object but not based on method].

30) What is class level lock when it is required?

Class level lock:

- Every class in java has a unique lock. If a Thread wants to execute a static synchronized method then it required class level lock.
- Once a Thread got class level lock then it is allow to execute any static synchronized method of that class.
- While a Thread executing any static synchronized method the remaining Threads are not allow to execute any static synchronized method of that class simultaneously.
- But remaining Threads are allowed to execute normal synchronized methods, normal static methods, and normal instance methods simultaneously.
- Class level lock and object lock both are different and there is no relationship between these two.

31) While a thread executing any synchronized method on the given object is it possible to execute remaining synchronized methods on the same object simultaneously by other thread?

- While a Thread executing any synchronized method the remaining Threads are not allowed execute any synchronized method on that object simultaneously. But remaining Threads are allowed to execute any non-synchronized method simultaneously. [Lock concept is implemented based on object but not based on method].

32) Difference between synchronized method and static synchronized method?

Synchronized method:

- If a Thread wants to execute any synchronized method on the given object 1st it has to get the lock of that object. Once a Thread got the lock of that object then it's allow to execute any synchronized method on that object. If the synchronized method execution completes then automatically Thread releases lock.

Static synchronized method:

- Every class in java has a unique lock. If a Thread wants to execute a static synchronized method then it required class level lock.
- Once a Thread got class level lock then it is allow to execute any static synchronized method of that class.
- While a Thread executing any static synchronized method the remaining Threads are not allow to execute any static synchronized method of that class simultaneously.
- But remaining Threads are allowed to execute normal synchronized methods, normal static methods, and normal instance methods simultaneously.

33) Advantages of synchronized block over synchronized method?

- If very few lines of the code required synchronization then it's never recommended to declare entire method as synchronized we have to enclose those few lines of the code with in synchronized block.
- The main advantage of synchronized block over synchronized method is it reduces waiting time of Thread and improves performance of the system.

34) What is synchronized statement?

- The statements which present inside synchronized method and synchronized block are called synchronized statements. [Interview people created terminology].

35) How two threads will communicate with each other?

- Two Threads can communicate with each other by using wait(), notify() and notifyAll() methods.
- The Thread which is excepting updation it has to call wait() method and the Thread which is performing updation it has to call notify() method. After getting notification the waiting Thread will get those updations.

36) Wait(),notify(),notifyAll(), are available in which class?

- wait(), notify() and notifyAll() methods are available in Object class but not in Thread class.

37) Why Wait(),notify(),notifyAll(), methods are defined in Object instead of Thread class?

- Because Thread can call these methods on any common object.

38) Without having the lock is it possible to call wait()?

- To call wait(), notify() and notifyAll() methods compulsory the current Thread should be owner of that object that is current Thread should has lock of that object that is current Thread should be in synchronized area. Hence we can call wait(), notify() and notifyAll() methods only from synchronized area otherwise we will get runtime exception saying IllegalMonitorStateException.

39) If a waiting thread gets notification then it will enter into which state?

40) In which methods thread can release lock?

- Except these (wait(),notify(),notifyAll()) methods there is no other place(method) where the lock release will be happen.

Method	Is Thread Releases Lock?
yield()	No
join()	No
sleep()	No
wait()	Yes
notify()	Yes
notifyAll()	Yes

- Once a Thread calls wait(), notify(), notifyAll() methods on any object then it releases the lock of that particular object but not all locks it has.

41) Explain Wait(),notify() and notifyAll() methods?

- Two Threads can communicate with each other by using wait(), notify() and notifyAll() methods.
- The Thread which is excepting updation it has to call wait() method and the Thread which is performing updation it has to call notify() method. After getting notification the waiting Thread will get those updations.

42) Difference between notify() and notifyAll() methods?

- We can use notify() method to give notification for only one Thread. If multiple Threads are waiting then only one Thread will get the chance and remaining Threads has to wait for further notification. But which Thread will be notify(inform) we can't expect exactly it depends on JVM.
- We can use notifyAll() method to give the notification for all waiting Threads. All waiting Threads will be notified and will be executed one by one.

43) Once a thread gives notification then which waiting thread will get a chance?

44) How a thread can interrupt another thread?

- We can interrupt a sleeping or waiting Thread by using interrupt()(break off) method of Thread class.

45) What is deadlock? Is it possible to resolve deadlock situation?

- If 2 Threads are waiting for each other forever(without end) such type of situation(infinite waiting) is called dead lock.
- There are no resolution techniques for dead lock but several prevention(avoidance) techniques are possible.

46) Which keyword causes deadlock situation?

- Synchronized keyword is the cause for deadlock hence whenever we are using synchronized keyword we have to take special care.

47) How we can stop a thread explicitly?

Step 1:

- Declare a boolean type variable and store false in that variable.

boolean stop=false;

Step 2:

- If the variable becomes true return from the run() method.

If(stop) return;

Step 3:

- Whenever to stop the Thread store true into the variable.

System.in.read();//press enter

Obj.stop=true;

48) How to kill a Thread in the middle of the line?

- We can call stop() method to stop a Thread in the middle then it will be entered into dead state immediately.

public final void stop();

- stop() method has been deprecated and hence not recommended to use.

49) Explain about suspend() and resume()?

- A Thread can suspend another Thread by using suspend() method then that Thread will be paused temporarily.
- A Thread can resume a suspended Thread by using resume() method then suspended Thread will continue its execution.

1) public final void suspend();

2) public final void resume();

- Both methods are deprecated and not recommended to use.

50) What is starvation and explain difference between deadlock and starvation?

- A long waiting of a Thread which never ends is called deadlock.
- A long waiting of a Thread which ends at certain point is called starvation.

51) What is race condition?

- Executing multiple Threads simultaneously and causing data inconsistency problems is nothing but Race condition we can resolve race condition by using synchronized keyword.

52) What is daemon thread? Give an example purpose of Daemon Thread?

Daemon Threads:

- The Threads which are executing in the background are called daemon Threads. The main objective of daemon Threads is to provide support for non daemon Threads.

Example:

Garbage collector

Example:

```
class MyThread extends Thread
{
        public void run()
        {
                for(int i=0;i<10;i++)
                {
                        System.out.println("lazy thread");
                        try
                        {
                                Thread.sleep(2000);
                        }
                        catch (InterruptedException e)
                        {}
                }
        }
}
class DaemonThreadDemo
{
        public static void main(String[] args)
        {
        MyThread t=new MyThread();
        t.setDaemon(true);----1---->
        t.start();
        System.out.println("end of main Thread");
        }
}
```

Output:

```
End of main Thread
```

Lazy thread

- If we are commenting line 1 then both main and child Threads are non daemon and hence both will be executed until they completion.
- If we are not commenting line 1 then main Thread is non daemon and child Thread is daemon and hence whenever main Thread terminates automatically child Thread will be terminated.

53) How we can check daemon nature of a thread? Is it possible to change daemon nature of a thread? Is main thread daemon (or) non-daemon?

- We can check whether the Thread is daemon or not by using isDaemon() method.

public final boolean isDaemon();

- We can change daemon nature of a Thread by using setDaemon () method.

public final void setDaemon(boolean b);

- But we can change daemon nature before starting Thread only. That is after starting the Thread if we are trying to change the daemon nature we will get R.E saying IllegalThreadStateException.
- Main Thread is always non daemon and we can't change its daemon nature because it's already started at the beginning only.
- Main Thread is always non daemon and for the remaining Threads daemon nature will be inheriting from parent to child that is if the parent is daemon child is also daemon and if the parent is non daemon then child is also non daemon.
- Whenever the last non daemon Thread terminates automatically all daemon Threads will be terminated.

Collections

1) What are limitations of object arrays?

Limitations of Object[] array:

1) Arrays are fixed in size that is once we created an array there is no chance of increasing (or) decreasing the size based on our requirement hence to use arrays concept compulsory we should know the size in advance which may not possible always.
2) Arrays can hold only homogeneous data elements.

Example:

```
Student[] s=new Student[10000];
s[0]=new Student();//valid
s[1]=new Customer();//invalid(compile time error)
```

Compile time error:

```
Test.java:7: cannot find symbol
Symbol: class Customer
Location: class Test
s[1]=new Customer();
```

3) But we can resolve this problem by using object type array(Object[]).

Example:

```
Object[] o=new Object[10000];
o[0]=new Student();
o[1]=new Customer();
```

4) Arrays concept is not implemented based on some data structure hence ready-made methods support we can't expert. For every requirement we have to write the code explicitly.

2) What are differences between arrays and collections?

Arrays	Collections
1) Arrays are fixed in size.	1) Collections are growable in nature.
2) Memory point of view arrays are not recommended to use.	2) Memory point of view collections are highly recommended to use.
3) Performance point of view arrays are recommended to use.	3) Performance point of view collections are not recommended to use.
4) Arrays can hold only homogeneous data type elements.	4) Collections can hold both homogeneous and heterogeneous elements.
5) There is no underlying data structure for arrays and hence there is no readymade method support.	5) Every collection class is implemented based on some standard data structure and hence readymade method support is available.
6) Arrays can hold both primitives and object types.	6) Collections can hold only objects but not primitives.

3) What are differences between arrays and ArrayList?

Arrays	ArrayList
1) Arrays are fixed in size.	1) ArrayListis growable in nature.
2) Memory point of view arrays are not recommended to use.	2) Memory point of view ArrayListis highly recommended to use.
3) Performance point of view arrays are recommended to use.	3) Performance point of view ArrayListis not recommended to use.
4) Arrays can hold only homogeneous data type elements.	4) ArrayList can hold both homogeneous and heterogeneous elements.

5) There is no underlying data structure for arrays and hence there is no readymade method support.	5) The underlying data structure is resizable array (or) growable array.
6) Arrays can hold both primitives and object types.	6) ArrayList can hold only objects but not primitives.

4) What are differences between arrays and Vector?

Arrays	Vector
1) Arrays are fixed in size.	1) Vectoris growable in nature.
2) Memory point of view arrays are not recommended to use.	2) Memory point of view Vectoris highly recommended to use.
3) Performance point of view arrays are recommended to use.	3) Performance point of view Vectoris not recommended to use.
4) Arrays can hold only homogeneous data type elements.	4) Vector can hold both homogeneous and heterogeneous elements.
5) There is no underlying data structure for arrays and hence there is no readymade method support.	5) The underlying data structure is resizable array (or) growable array.
6) Arrays can hold bothtives and object types.	6) Vector can hold only objects but not primitives.

5) What is collection API?

Collection: If we want to represent a group of objects as a single entity then we should go for collection.

6) What is collection framework?

Collection framework: It defines several classes and interfaces to represent a group of objects as a single entity.

7) What is difference between Collections and Collection?

- "Collection is an "interface" which can be used to represent a group of objects as a single entity. Whereas "Collections is an utility class" present in java.util package to define several utility methods for Collection objects.

Collection----------interface

Collections----------class

8) Explain about Collection interface?

Collection:

1) If we want to represent a group of "individual objects" as a single entity then we should go for collection.
2) In general we can consider collection as root interface of entire collection framework.

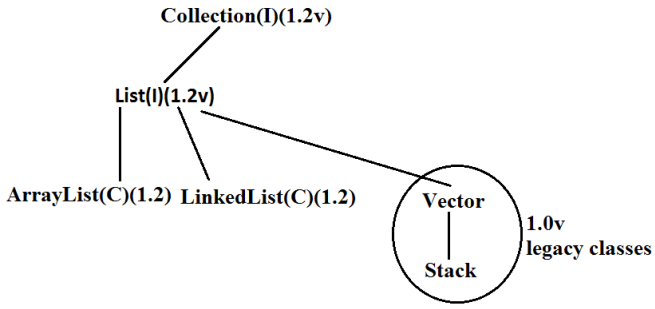
3) Collection interface defines the most common methods which can be applicable for any collection object.

9) Explain about List Interface?

List interface:

- It is the child interface of Collection.
- If we want to represent a group of individual objects where duplicates are allow and insertion order is preserved. Then we should go for List.
- We can differentiate duplicate objects and we can maintain insertion order by means of index hence "index play very important role in List".

Diagram:



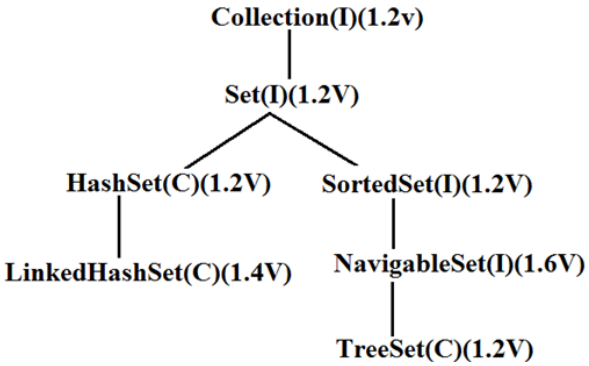
- Vector and Stack classes are reengineered in 1.2 versions to implement List interface.

10) Explain about Set Interface?

Set interface:

1) It is the child interface of Collection.
2) If we want to represent a group of individual objects where duplicates are not allow and insertion order is not preserved then we should go for Set interface.

Diagram:



- Set interface does not contain any new method we have to use only Collection interface methods.

11) Explain about SortedSet Interface?

SortedSet:

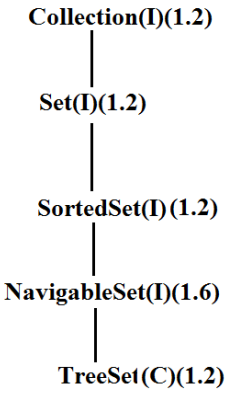
1) It is child interface of Set.
2) If we want to represent a group of "unique objects" according to some sorting order then we should go for SortedSet interface.
3) That sorting order can be either default natural sorting (or) customized sorting order.

12) Explain about NavigableSet?

NavigableSet:

1) It is the child interface of SortedSet.
2) It provides several methods for navigation purposes.

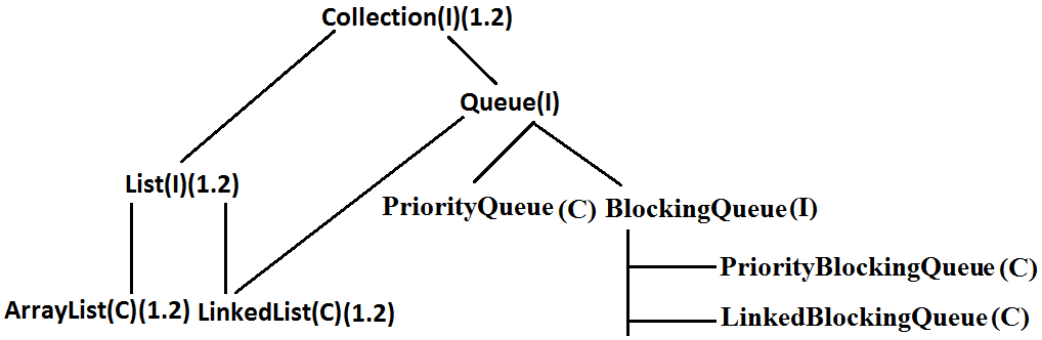
Diagram:



13) Explain about Queue Interface?

1) If we want to represent a group of individual objects prior (happening before something else) to processing then we should go for Queue interface.

Diagram:



2) Usually Queue follows **first in first out order** but based on our requirement we can implement our own order also.
3) From 1.5v onwards LinkedList also implements Queue interface.

14) Explain about Map Interface?

Map:

1) If we want to represent a group of objects as "key-value" pair then we should go for Map interface.
2) Both key and value are objects only.
3) Duplicate keys are not allowed but values can be duplicated
4) Each key-value pair is called "one entry".

Diagram:

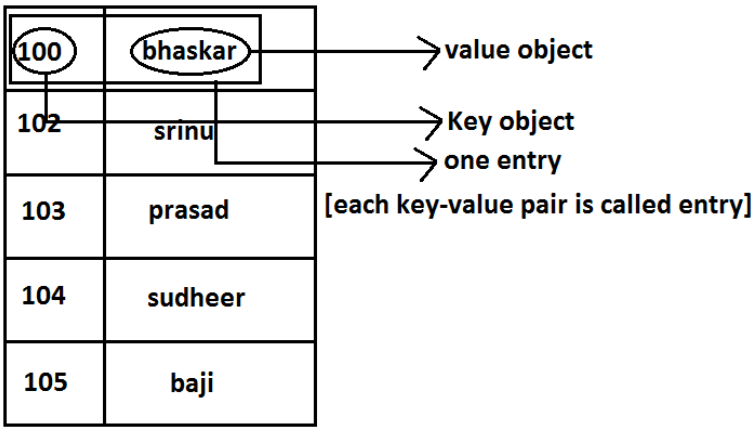
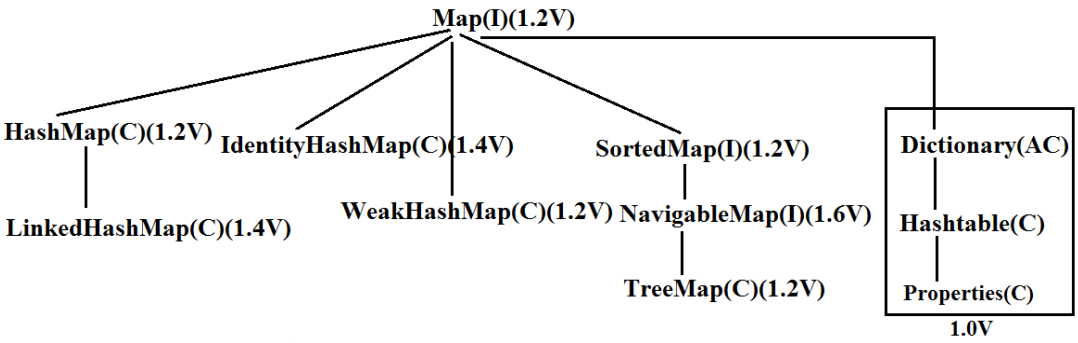


Diagram:



- Map interface is not child interface of Collection and hence we can't apply Collection interface methods here.
- Map interface defines the specific methods.

15) Explain about SortedMap?

SortedMap:

- It is the child interface of Map.
- If we want to represent a group of key-value pairs according to some sorting order of keys then we should go for SortedMap.

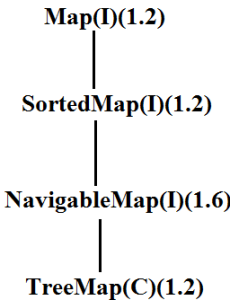
- Sorting is possible only based on the keys but not based on values.

16) Explain about NavigableMap?

NavigableMap:

- It is the child interface of SortedMap and it defines several methods for navigation purpose.

Diagram:



17) Explain about ArrayList class?

ArrayList:

1) The underlying data structure is resizable array (or) growable array.
2) Duplicate objects are allowed.
3) Insertion order preserved.
4) Heterogeneous objects are allowed.
5) Null insertion is possible.

18) What is RandomAccess Interface?

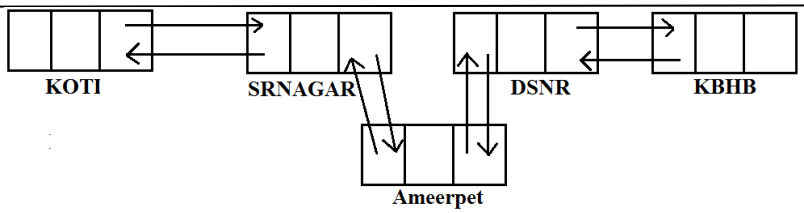
- RandomAccess interface present in util package and doesn't contain any methods. It is a marker interface.

19) Explain about LinkedList Class?

LinkedList:

1) The underlying data structure is double LinkedList
2) If our frequent operation is insertion (or) deletion in the middle then LinkedList is the best choice.
3) If our frequent operation is retrieval operation then LinkedList is worst choice.
4) Duplicate objects are allowed.
5) Insertion order is preserved.
6) Heterogeneous objects are allowed.
7) Null insertion is possible.
8) Implements Serializable and Cloneable interfaces but not RandomAccess.

Diagram:



- Usually we can use LinkedList to implement Stacks and Queues to provide support for this requirement

20) Explain about Vector Class?

Vector:

1) The underlying data structure is resizable array (or) growable array.
2) Duplicate objects are allowed.
3) Insertion order is preserved.
4) Heterogeneous objects are allowed.
5) Null insertion is possible.
6) Implements Serializable, Cloneable and RandomAccess interfaces.

- Every method present in Vector is synchronized and hence Vector is Thread safe.

21) What is difference between ArrayList and Vector?

ArrayList	Vector
1) No method is synchronized	1) Every method is synchronized
2) At a time multiple Threads are allow to operate on ArrayList object and hence ArrayList object is not Thread safe.	2) At a time only one Thread is allow to operate on Vector object and hence Vector object is Thread safe.
3) Relatively performance is high because Threads are not required to wait.	3) Relatively performance is low because Threads are required to wait.
4) It is non legacy and introduced in 1.2v	4) It is legacy and introduced in 1.0v

22) How we can get synchronized version of ArrayList?

Getting synchronized version of ArrayList object:

- Collections class defines the following method to return synchronized version of List.

Public static List synchronizedList(list l);

Example:

```
ArrayList a=new arrayList();
List l1=collections.synchronizedList(a);
```

synchronized version

nonsynchronized version

- Similarly we can get synchronized version of Set and Map objects by using the following methods.
    1) **public static Set synchronizedSet(Set s);**
    2) **public static Map synchronizedMap(Map m);**

23) What is difference between size and capacity of a Collection Object?

Size: Size means how many objects are added to the collection object.

Capacity: Capacity means total capacity of the collection object.

24) What is difference between ArrayList and LinkedList?

ArrayList	LinkedList
1) The underlying data structure is resizable array (or) growable array.	9) The underlying data structure is double LinkedList.
• ArrayList is the best choice if our frequent operation is retrieval.	10) LinkedListis the best choice if our frequent operation is insertion (or) deletion in the middle.

25) What are legacy classes and interfaces present in Collections framework?

1) Enumeration(I)
2) Dictionary(AC)
3) Vector(C)
4) Stack(C)
5) Hashtable(C)
6) Properties(C)

26) What is difference between Enumeration and Iterator?

Property	Enumeration	Iterator	ListIterator
1) Is it legacy?	yes	No	No
2) It is applicable for?	Only legacy classes.	Applicable for any collection object.	Applicable for only list objects.
3) Moment?	Single direction cursor(forward)	Single direction cursor(forward	Bi-directional.
4) How to get it?	By usingelements() method.	By using iterator()method.	By using listIterator() method.
5) Accessibility?	Only read.	Both read and	Read/remove/replace/add.

		remove.	
6) methods	hasMoreElement() nextElement()	hasNext() next() remove()	9 methods.

27) What are limitations of Enumeration?

Limitations of Enumeration:

1) We can apply Enumeration concept only for legacy classes and it is not a universal cursor.
2) By using Enumeration we can get only read access and we can't perform remove operations.
3) To overcome these limitations sun people introduced Iterator concept in 1.2v.

28) What is difference between Enum and Enumeration?

- We can use Enum to define a group of named constants.

Example 1:

```
enum Month
{
JAN,FEB,MAR,DEC;
}
```

Example 2:

```
enum Beer
{
KF,KO,RC,FO;
}
```

- Enum concept introduced in 1.5 versions.g33333.1
- When compared with old languages Enum java's Enum is more powerful.
- By using Enum we can define our own data types which are also come enumerated data types.

Enumeration:

1) We can use Enumeration to get objects one by one from the legacy collection objects.
2) We can create Enumeration object by using elements() method.

Enumeration e=v.elements();

Vector Object

- Enumeration interface defines the following two methods
  1) **public boolean hasMoreElements();**
  2) **public Object nextElement();**

29) What is relation between ListIterator and Iterator?

1) ListIterator is the child interface of Iterator.

2) By using listIterator we can move either to the forward direction (or) to the backward direction that is it is a bi-directional cursor.
3) While iterating by listIterator we can perform replacement and addition of new objects in addition to read and remove operations

- By using listIterator method we can create listIterator object.

ListIterator itr=l.listiterator();

30) Explain about HashSet class?

HashSet:

1) The underlying data structure is Hashtable.
2) Insertion order is not preserved and it is based on hash code of the objects.
3) Duplicate objects are not allowed.
4) If we are trying to insert duplicate objects we won't get compile time error and runtime error add() method simply returns false.
5) Heterogeneous objects are allowed.
6) Null insertion is possible.
7) Implements Serializable and Cloneable interfaces but not RandomAccess.

31) If we are trying to insert duplicate values in Set what will happen?

- If we are trying to insert duplicate objects we won't get any compile time error and runtime error add() method simply returns false.

32) What is LinkedHashSet?

LinkedHashSet:

1) It is the child class of HashSet.
2) LinkedHashSet is exactly same as HashSet except the following differences.

HashSet	LinkedHashSet
1) The underlying data structure is Hashtable.	1) The underlying data structure is a combination of LinkedList and Hashtable.
2) Insertion order is not preserved.	2) Insertion order is preserved.
3) Introduced in 1.2 v.	3) Introduced in 1.4v.

33) What is difference between HashSet and LinkedHashSet?

HashSet	LinkedHashSet
4) The underlying data structure is Hashtable.	4) The underlying data structure is a combination of LinkedList and Hashtable.
5) Insertion order is not preserved.	5) Insertion order is preserved.
6) Introduced in 1.2 v.	6) Introduced in 1.4v.

34) What are major enhancements in 1.4 version of collection framework?

1) LinkedHashSet

2) LinkedHashMap
3) IdentityHashMap

35) Explain about TreeSet?

TreeSet:

1) The underlying data structure is balanced tree.
2) Duplicate objects are not allowed.
3) Insertion order is not preserved and it is based on some sorting order of objects.
4) Heterogeneous objects are not allowed if we are trying to insert heterogeneous objects then we will get ClassCastException.
5) Null insertion is possible(only once).

36) What is difference between List and Set interfaces?

Set	List
1) It is the child interface of Collection.	1) It is the child interface of Collection.
2) If we want to represent a group of individual objects where duplicates are not allow and insertion order is not preserved then we should go for Set interface.	2) If we want to represent a group of individual objects where duplicates are allow and insertion order is preserved. Then we should go for List.

37) What is Comparable interface?

Comparable interface:

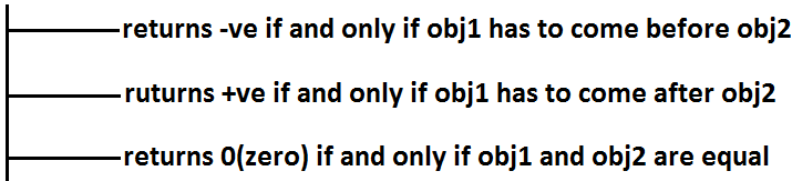
- Comparable interface present in java.lang package and contains only one method compareTo() method.

public int compareTo(Object obj);

Example:

obj1.compareTo(obj2);

Diagram:



Example 3:

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println("A".compareTo("Z"));//-25
                System.out.println("Z".compareTo("K"));//15
```

```
                System.out.println("A".compareTo("A"));//0
                //System.out.println("A".compareTo(new Integer(10)));//Test.java:8:
compareTo(java.lang.String) in java.lang.String cannot be applied to (java.lang.Integer)
                //System.out.println("A".compareTo(null));//NullPointerException
        }
}
```

- If we are depending on default natural sorting order then internally JVM will use compareTo() method to arrange objects in sorting order.

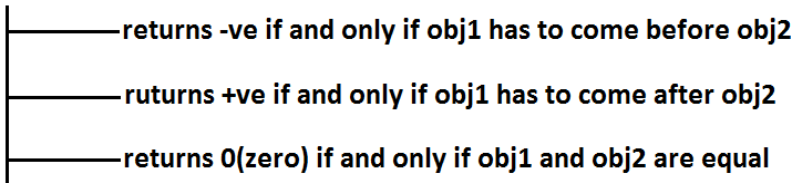
38) What is Comparator interface?

Comparator interface:

- Comparator interface present in java.util package this interface defines the following 2 methods.

1) public int compare(Object obj1,Object Obj2);

Diagram:



2) **public boolean equals(Objectobj);**

- Whenever we are implementing Comparator interface we have to provide implementation only for **compare()** method.
- Implementing equals() method is optional because it is already available from Object class through inheritance.

Requirement: Write a program to insert integer objects into the TreeSet where the sorting order is descending order.

Program:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                TreeSet t=new TreeSet(new MyComparator());----------(1)
                t.add(10);
                t.add(0);
                t.add(15);
                t.add(5);
                t.add(20);
                System.out.println(t);//[20, 15, 10, 5, 0]
```

```
        }
}
class MyComparator implements Comparator
{
        public int compare(Object obj1,Object obj2)
        {
                Integer i1=(Integer)obj1;
                Integer i2=(Integer)obj2;
                if(i1<i2)
                        return +1;
                else if(i1>i2)
                        return -100;
                else return 0;
        }}
```

- At line "1" if we are not passing Comparator object then JVM will always calls compareTo() method which is meant for default natural sorting order(ascending order)hence in this case the output is [0, 5, 10, 15, 20].
- At line "1" if we are passing Comparator object then JVM calls compare() method of MyComparator class which is meant for customized sorting order(descending order) hence in this case the output is [20, 15, 10, 5, 0].

Note: Whenever we are defining our own customized sorting by Comparator then the objects need not be Comparable.

- **Example:** StringBuffer

Note: If we are depending on default natural sorting order then the objects should be "homogeneous and comparable" otherwise we will get ClassCastException. If we are defining our own sorting by Comparator then objects "need not be homogeneous and comparable".

- Comparable meant for default natural sorting order.
- Comparator meant for customized sorting order.

39) What is difference between Comparable and Comparator?

Comparable	Comparator
1) Comparable meant for default natural sorting order.	1) Comparator meant for customized sorting order.
2) Present in java.lang package.	2) Present in java.util package.
3) Contains only one method. compareTo() method.	3) Contains 2 methods. Compare() method. Equals() method.
4) String class and all wrapper	4) No predefined class implements

Classes implements Comparable interface.	Comparator.

40) What is difference between HashSet and TreeSet?

Property	HashSet	LinkedHashSet	TreeSet
1) Underlying Data structure.	Hashtable.	LinkedList +Hashtable.	Balanced Tree.
2) Insertion order.	Not preserved.	Preserved.	Not preserved (by default).
3) Duplicate objects.	Not allowed.	Not allowed.	Not allowed.
4) Sorting order.	Not applicable.	Not applicable.	Applicable.
5) Heterogeneous objects.	Allowed.	Allowed.	Not allowed.
6) Null insertion.	Allowed.	Allowed.	For the empty TreeSet as the 1st element null insertion is possible in all other cases we will get NPE.

41) What is Entry interface?

Entry interface:

- Each key-value pair is called one entry without existing Map object there is no chance of existing entry object hence interface entry is define inside Map interface(inner interface).

Example:

```
interface Map
{
        .................;
        .................;
        .................;
        interface Entry
        {
                Object getKey();
                Object getValue();              on Entry we can apply these 3 methods.
                Object setValue(Object new);
        }
}
```

42) Explain about HashMap?

HashMap:

1) The underlying data structure is Hashtable.
2) Duplicate keys are not allowed but values can be duplicated.
3) Insertion order is not preserved and it is based on hash code of the keys.
4) Heterogeneous objects are allowed for both key and value.
5) Null is allowed for keys(only once) and for values(any number).

43) Explain about LinkedHashMap?

1) It is the child class of HashMap.
2) LinkedHashMap is exactly same as HashMap except the following differences.

HashMap	LinkedHashMap
1) The underlying data structure is Hashtable.	1) The underlying data structure is a combination of Hashtable+ LinkedList.
2) Insertion order is not preserved.	2) Insertion order is preserved.
3) introduced in 1.2.v.	3) Introduced in 1.4v.

Note: In general we can use LinkedHashSet and LinkedHashMap for implementing cache applications.

44) What is the difference between HashMap and LinkedHashMap?

HashMap	LinkedHashMap
4) The underlying data structure is Hashtable.	4) The underlying data structure is a combination of Hashtable+ LinkedList.
5) Insertion order is not preserved.	5) Insertion order is preserved.
6) introduced in 1.2.v.	6) Introduced in 1.4v.

45) What is the difference between HashMap and Hashtable?

HashMap	Hashtable
1) No method is synchronized.	1) Every method is synchronized.
2) Multiple Threads can operate simultaneously on HashMap object and hence it is not Thread safe.	2) Multiple Threads can't operate simultaneously on Hashtable object and hence Hashtable object is Thread safe.
3) Relatively performance is high.	3) Relatively performance is low.
4) Null is allowed for both key and value.	4) Null is not allowed for both key and value otherwise we will get NullPointerException.
5) It is non legacy and introduced in 1.2v.	5) It is legacy and introduced in 1.0v.

46) What is IdentityHashMap?

1) The underlying data structure is Hashtable.
2) Duplicate keys are not allowed but values can be duplicated.
3) Insertion order is not preserved and it is based on hash code of the keys.
4) Heterogeneous objects are allowed for both key and value.
5) Null is allowed for keys(only once) and for values(any number).

47) What is difference between HashMap and IdentityHashMap?

1) It is exactly same as HashMap except the following differences.
2) In the case of HashMap JVM will always use ".equals()"method to identify duplicate keys.
3) But in the case of IdentityHashMap JVM will use== (double equal operator) to identify duplicate keys.

Example:

```
import java.util.*;
class HashMapDemo
{
        public static void main(String[] args)
        {
                HashMap m=new HashMap();
                Integer i1=new Integer(10);
                Integer i2=new Integer(10);
                m.put(i1,"pavan");
                m.put(i2,"kalyan");
                System.out.println(m);
        }}
```

- In the above program i1 and i2 are duplicate keys because i1.equals(i2) returns true.
- In the above program if we replace HashMap with IdentityHashMap then i1 and i2 are not duplicate keys because i1==i2 is false hence in this case the output is {10=pavan, 10=kalyan}.

```
System.out.println(m.get(10));//null
10==i1------false
10==i2------false
```

48) What is WeakHashMap?

1) The underlying data structure is Hashtable.
2) Duplicate keys are not allowed but values can be duplicated.
3) Insertion order is not preserved and it is based on hash code of the keys.
4) Heterogeneous objects are allowed for both key and value.
5) Null is allowed for keys(only once) and for values(any number).

49) What is difference between HashMap and WeakHashMap?

- It is exactly same as HashMap except the following differences.
- In the case of normal HashMap, an object is not eligible for GC even though it doesn't have any references if it is associated with HashMap. That is HashMap dominates garbage collector.
- But in the case of WeakHashMap if an object does not have any references then it's always eligible for GC even though it is associated with WeakHashMap that is garbage collector dominates WeakHashMap.

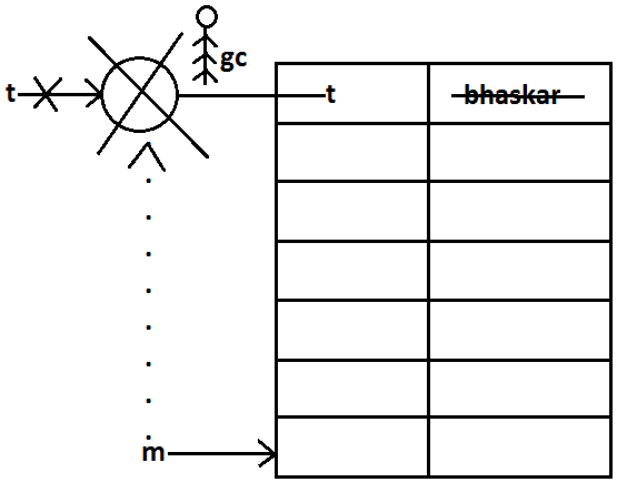
Example:

```
import java.util.*;
class WeakHashMapDemo
{
	public static void main(String[] args)throws Exception
	{
		WeakHashMap m=new WeakHashMap();
		Temp t=new Temp();
		m.put(t,"bhaskar");
		System.out.println(m);//{Temp=bhaskar}
		t=null;
		System.gc();
		Thread.sleep(5000);
		System.out.println(m);//{}
	}
}
class Temp
{
	public String toString()
	{
		return "Temp";
	}
	public void finalize()
	{
		System.out.println("finalize() method called");
	}}
```

Output:

```
{Temp=bhaskar}
finalize() method called
{}
```

Diagram:



- In the above program if we replace WeakHashMap with normal HashMap then object won't be destroyed by the garbage collector in this the output is

{Temp=bhaskar}
{Temp=bhaskar}

50) What is TreeMap?

TreeMap:

1) The underlying data structure is RED-BLACK Tree.
2) Duplicate keys are not allowed but values can be duplicated.
3) Insertion order is not preserved and all entries will be inserted according to some sorting order of keys.
4) If we are depending on default natural sorting order keys should be homogeneous and Comparable otherwise we will get ClassCastException.
5) If we are defining our own sorting order by Comparator then keys can be heterogeneous and non Comparable.
6) There are no restrictions on values they can be heterogeneous and non Comparable.
7) For the empty TreeMap as first entry null key is allowed but after inserting that entry if we are trying to insert any other entry we will get NullPointerException.
8) For the non empty TreeMap if we are trying to insert an entry with null key we will get NullPointerException.
9) There are no restrictions for null values.

Example:

```
import java.util.*;
class TreeMapDemo
{
        public static void main(String[] args)
        {
                TreeMap t=new TreeMap();
```

```
                t.put(100,"ZZZ");
                t.put(103,"YYY");
                t.put(101,"XXX");
                t.put(104,106);
                t.put(107,null);
                //t.put("FFF","XXX");//ClassCastException
                //t.put(null,"xxx");//NullPointerException
                System.out.println(t);//{100=ZZZ, 101=XXX, 103=YYY, 104=106, 107=null}
        }
}
```

51) What is Hashtable?

Hashtable:

1) The underlying data structure is Hashtable.
2) Insertion order is not preserved and it is based on hash code of the keys.
3) Heterogeneous objects are allowed for both keys and values.
4) Null key (or) null value is not allowed otherwise we will get NullPointerException.
5) Duplicate keys are allowed but values can be duplicated.

52) What is PriorityQueue?

PriorityQueue:

1) We can use PriorityQueue to represent a group of individual objects prior to processing according to some priority.
2) The priority order can be either default natural sorting order (or) customized sorting order specified by Comparator object.
3) If we are depending on default natural sorting order then the objects must be homogeneous and Comparable otherwise we will get ClassCastException.
4) If we are defining our own customized sorting order by Comparator then the objects need not be homogeneous and Comparable.
5) Duplicate objects are not allowed.
6) Insertion order is not preserved but all objects will be inserted according to some priority.
7) Null is not allowed even as the 1st element for empty PriorityQueue.

Note: Some platforms may not provide proper supports for PriorityQueue [windowsXP].

53) What is Arrays class?

Arrays class:

- Arrays class defines several utility methods for arrays.

Sorting the elements of array:

public static void sort(primitive[] p);//any primitive data type we can give

- To sort the elements of primitive array according to default natural sorting order.

public static void sort(object[] o);

- To sort the elements of object[] array according to default natural sorting order.
- In this case objects should be homogeneous and Comparable.

public static void sort(object[] o,Comparator c);

- To sort the elements of object[] array according to customized sorting order.

Note: We can sort object[] array either by default natural sorting order (or) customized sorting order but we can sort primitive arrays only by default natural sorting order.

54) We are planning to do an indexed search in a list of objects. Which of the two java collections should you use: ArrayList (or) LinkedList?

- ArrayList is the best choice if our frequent operation is retrieval.

55) Why ArrayList is faster than Vector?

ArrayList:

- Relatively performance is high because Threads are not required to wait.

Vector:

- Relatively performance is low because Threads are required to wait.

56) How we can sort the elements of list?

Sorting the elements of a List:

- Collections class defines the following methods to perform sorting the elements of a List.

public static void sort(List l);

- To sort the elements of List according to default natural sorting order in this case the elements should be homogeneous and comparable otherwise we will get ClassCastException.
- The List should not contain null otherwise we will get NullPointerException.

public static void sort(List l,Comparator c);

- To sort the elements of List according to customized sorting order.

Program 1: To sort elements of List according to natural sorting order.

```
import java.util.*;
class CollectionsDemo
{
        public static void main(String[] args)
        {
                ArrayList l=new ArrayList();
                l.add("Z");
                l.add("A");
                l.add("K");
                l.add("N");
                //l.add(new Integer(10));//ClassCastException
```

```
                //l.add(null);//NullPointerException
                System.out.println("Before sorting :"+l);//[Z, A, K, N]
                Collections.sort(l);
                System.out.println("After sorting :"+l);//[A, K, N, Z]
        }
}
```

Program 2: To sort elements of List according to customized sorting order.

```
import java.util.*;
class CollectionsDemo
{
        public static void main(String[] args)
        {
                ArrayList l=new ArrayList();
                l.add("Z");
                l.add("A");
                l.add("K");
                l.add("L");
                l.add(new Integer(10));
                //l.add(null);//NullPointerException
                System.out.println("Before sorting :"+l);//[Z, A, K, L, 10]
                Collections.sort(l,new MyComparator());
                System.out.println("After sorting :"+l);//[Z, L, K, A, 10]
                }
}
class MyComparator implements Comparator
{
        public int compare(Object obj1,Object obj2)
        {
                String s1=(String)obj1;
                String s2=obj2.toString();
                return s2.compareTo(s1);
        }
}
```

57) How we can reverse the order of elements of a list?

Reversing the elements of List:

public static void reverse(List l);

Program: To reverse elements of list.

```
import java.util.*;
```

```
class CollectionsReverseDemo
{
	public static void main(String[] args)
	{
		ArrayList l=new ArrayList();
		l.add(15);
		l.add(0);
		l.add(20);
		l.add(10);
		l.add(5);
		System.out.println(l);//[15, 0, 20, 10, 5]
		Collections.reverse(l);
		System.out.println(l);//[5, 10, 20, 0, 15]
	}}
```

58) Explain about Collections class?

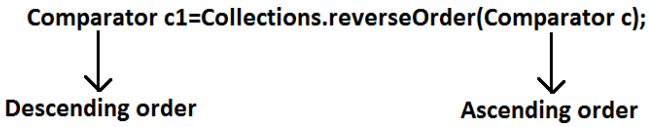
Collections class:

- Collections class defines several utility methods for collection objects.

59) What is the difference between reverse() and reverseOrder() present in Collections class?

reverse() vs reverseOrder() method

- We can use reverse() method to reverse the elements of List.
- Where as we can use reverseOrder() method to get reversed Comparator.



60) Explain about arrays class?

Arrays class:

- Arrays class defines several utility methods for arrays.

61) How we can implement sorting for arrays?

Sorting the elements of array:

public static void sort(primitive[] p);//any primitive data type we can give

- To sort the elements of primitive array according to default natural sorting order.

public static void sort(object[] o);

- To sort the elements of object[] array according to default natural sorting order.
- In this case objects should be homogeneous and Comparable.

public static void sort(object[] o,Comparator c);

- To sort the elements of object[] array according to customized sorting order.

Note: We can sort object[] array either by default natural sorting order (or) customized sorting order but we can sort primitive arrays only by default natural sorting order.

Program: To sort elements of array.

```
import java.util.*;
class ArraySortDemo
{
	public static void main(String[] args)
	{
		int[] a={10,5,20,11,6};
		System.out.println("primitive array before sorting");
		for(int a1:a)
		{
			System.out.println(a1);
		}
		Arrays.sort(a);
		System.out.println("primitive array after sorting");
		for(int a1: a)
		{
			System.out.println(a1);
		}
		String[] s={"A","Z","B"};
		System.out.println("Object array before sorting");
		for(String s1: s)
		{
			System.out.println(s1);
		}
		Arrays.sort(s);
		System.out.println("Object array after sorting");
		for(String s1:s)
		{
			System.out.println(s1);
		}
		Arrays.sort(s,new MyComparator());
		System.out.println("Object array after sorting by Comparator:");
		for(String s1: s)
		{
			System.out.println(s1);
		}
```

```
        }
}
class MyComparator implements Comparator
{
        public int compare(Object obj1,Object obj2)
        {
                String s1=obj1.toString();
                String s2=obj2.toString();
                return s2.compareTo(s1);
        }
}
```

62) Is it possible to convert Collection object to array?

- Yes, we can convert Collection object to array by using toArray() method in the ArrayList class.

63) Is it possible to convert array into ArrayList form?

- Yes, we can convert array to ArrayList form by using asList() method of Arrays class.

64) Explain about binarySearch() method available in Collections and Arrays classes?

Collections:

- Collections class defines the following methods to search the elements of a List.

public static int binarySearch(List l,Object obj);

- If the List is sorted according to default natural sorting order then we have to use this method.

public static int binarySearch(List l,Object obj,Comparator c);

- If the List is sorted according to Comparator then we have to use this method.

Arrays:

- Arrays class defines the following methods to search elements of array.

1) **public static int binarySearch(primitive[] p,primitive key);**
2) **public static int binarySearch(Object[] p, object key);**
3) **public static int binarySearch(Object[] p,Object key,Comparator c);**

- All rules of Arrays class binarySearch() method are exactly same as Collections class binarySearch() method.

Program: To search elements of array.

```
import java.util.*;
class ArraysSearchDemo
{
        public static void main(String[] args)
        {
                int[] a={10,5,20,11,6};
```

```
        Arrays.sort(a);
        System.out.println(Arrays.binarySearch(a,6));//1
        System.out.println(Arrays.binarySearch(a,14));//-5
        String[] s={"A","Z","B"};
        Arrays.sort(s);
        System.out.println(Arrays.binarySearch(s,"Z"));//2
        System.out.println(Arrays.binarySearch(s,"S"));//-3
        Arrays.sort(s,new MyComparator());
        System.out.println(Arrays.binarySearch(s,"Z",new MyComparator()));//0
        System.out.println(Arrays.binarySearch(s,"S",new MyComparator()));//-2
        System.out.println(Arrays.binarySearch(s,"N"));//-4(unpredictable result)
        }
}
```

65) What is the difference between index and insertion point?

- Successful search returns **index** unsuccessful search returns **insertion point**.

66) What is load-factor?

- Creates an empty HashSet object with default initial capacity 16 and default fill ratio 0.75(fill ratio is also known as load factor).

Java.Lang.Object Class

1) What is your favorite package?

- The most commonly required classes and interfaces are encapsulated in the separate package which is nothing but java.lang package.
- It is not required to import java.lang package in our program because it is available by default to every java program.

2) What is your favorite class?

Java.Lang.Object class: For any java object whether it is predefine or customized the most commonly required methods are encapsulated into a separate class which is nothing but object class.

- As object class acts as a root (or) parent (or) super for all java classes, by default its methods are available to every java class.

3) Whenever we are printing object reference which method will be execute?

- Whenever we are try to print any object reference internally toString() method will be executed.
- If our class doesn't contain toString() method then Object class toString() method will be executed.

Example: System.out.println(s1); ➔super(s1.toString());

4) In Object class how the toString() method is implemented?

```
public String toString() {
        return getClass().getName() + "@" + Integer.toHexString(hashCode());
  }
```

5) Purpose of toString() method?

- We can use toString() method to get string representation of an object.

6) What is hashCode of an object? Where it will be useful?

- For every object JVM will generate a unique number which is nothing but hashCode.
- hashCode of an object will be used by JVM while saving objects into HashSet, HashMap, and Hashtable etc.
- If the objects are stored according to hashCode searching will become very efficient (The most powerful search algorithm is hashing which will work based on hashCode).

7) Is it possible to override hashCode() method in our class?

- Based on our programming requirement we can override hashCode() method in our class.

8) Relation between toString() and hashCode()?

- If we are giving Opturnuty to Object class toString() method it internally calls hashCode() method. But if we are overriding toString() method it may not call hashCode() method.

- We can use toString() method while printing object references and we can use hashCode() method while saving objects into HashSet or Hashtable or HashMap.

9) Does hashCode represent address of an object?

- If we didn't override hashCode() method then Object class hashCode() method will be executed which generates hashCode based on address of the object but it doesn't mean hashCode represents address of the object.

10) What is the purpose of .equals() method?

- We can use this method to check equivalence of two objects.

11) In Object class how .equals() is implemented?

- If our class doesn't contain .equals() method then object class .equals() method will be executed which is always meant for reference compression[address compression].

12) Is it possible to override .equals() method?

- Yes it is possible to override .equals() method.

13) Relations between == operator and .equals()?

Relationship between .equals() method and ==(double equal operator):

1) If r1==r2 is true then r1.equals(r2) is always true that is if two objects are equal by double equal operator then these objects are always equal by .equals() method also.
2) If r1==r2 is false then we can't conclude anything about r1.equals(r2) it may return true (or) false.
3) If r1.equals(r2) is true then we can't conclude anything about r1==r2 it may returns true (or) false.
4) If r1.equals(r2) is false then r1==r2 is always false.

14) Difference between == and .equals()?

==(double equal operator)	.equals() method
1) It is an operator applicable for both primitives and object references.	1) It is a method applicable only for object references but not for primitives.
2) In the case of primitives == (double equal operator) meant for content compression, but in the case of object references == operator meant for reference compression.	2) By default .equals() method present in object class is also meant for reference compression.
3) We can't override== operator for content compression in object references.	3) We can override .equals() method for content compression.
4) If there is no relationship between argument types then we will get compile time error saying incompatible	4) If there is no relationship between argument types then .equals() method simply returns false and we won't get

types.	any compile time error and runtime error.
5) For any object reference r, r==null is always false.	5) For any object reference r, r.equals(null) is also returns false.

Note: In general we can use == (double equal operator) for reference compression whereas .equals() method for content compression.

15) What is contract between .equals() and hashCode()?

Contract between .equals() method and hashCode() method:

1) If 2 objects are equal by .equals() method compulsory their hashcodes must be equal (or) same. That is
   If r1.equals(r2) is true then r1.hascode()==r2.hashcode() must be true.
2) If 2 objects are not equal by .equals() method then there are no restrictions on hashCode() methods. They may be same (or) may be different. That is
   If r1.equals(r2) is false then r1.hashCode()==r2.hashCode() may be same (or) may be different.
3) If hashcodes of 2 objects are equal we can't conclude anything about .equals() method it may returns true (or) false. That is
   If r1.hashCode()==r2.hashCode() is true then r1.equals(r2) method may returns true (or) false.
4) If hashcodes of 2 objects are not equal then these objects are always not equal by .equals() method also. That is
   If r1.hashCode()==r2.hashCode() is false then r1.equals(r2) is always false.

- To maintain the above contract between .equals() and hashCode() methods whenever we are overriding .equals() method compulsory we should override hashCode() method. Violation leads to no compile time error and runtime error but it is not good programming practice.
- Consider the following person class.

16) In which case we can override .equals()?

- Based on our programming requirement we can override .equals() method for content compression purpose.

17) Whenever we are overriding .equals() which method we have to override?

- Object class .equals() method we have to override.

18) What is cloning? How we can perform cloning?

- The process of creating exactly duplicate object is called cloning.
- The main objective of cloning is to maintain backup.
- That is if something goes wrong we can recover the situation by using backup copy.
- We can perform cloning by using clone() method of Object class.

protected native object clone() throws CloneNotSupportedException;

19) What is Cloneable interface?

- An object is said to be Cloneable if and only if the corresponding class implements Cloneable interface.
- Cloneable interface present in java.lang package and does not contain any methods. It is a marker interface where the required ability will be provided automatically by the JVM.

20) Proper way of overriding hashCode()?

- Overriding hashCode() method is said to be proper if and only if for every object we have to generate a unique number.

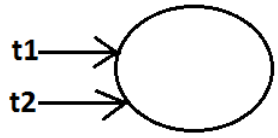
21) Difference between deepcloning and shallowcloning?

- The process of creating just duplicate reference variable but not duplicate object is called shallow cloning.

Example:

```
Test t1=new Test();
Test t2=t1;
```

Diagram:

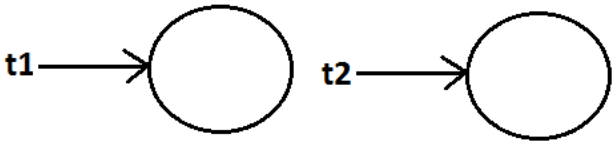


- The process of creating exactly independent duplicate object is called deep cloning.

Example:

```
Test t1=new Test();
Test t2=(Test)t1.clone();
System.out.println(t1==t2);//false
System.out.println(t1.hashCode()==t2.hashCode());//false
```

Diagram:



- Cloning by default deep cloning.

22) In your previous project where you used the following method toString(),HashCode(),equals(),clone()?

23) What are various methods present in java.lang.Object?

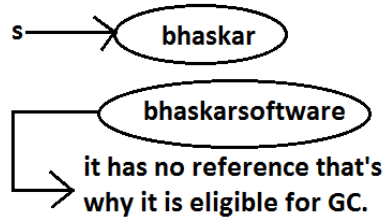
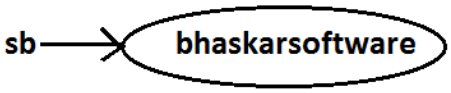
- The following is the list of all methods present in java.lang Object class.
    1) public StringtoString();
    2) public native int hashCode();
    3) public boolean equals(Object o);
    4) protected native Objectclone()throws CloneNotSupportedException;

5) public final Class<?>getClass();
6) protected void finalize()throws Throwable;
7) public final void wait()throws InterruptedException;
8) public final native void wait()throws InterruptedException;
9) public final void wait(long ms,int ns)throws InterruptedException;
10) public final native void notify();
11) public final native void notifyAll();

String, StringBuffer and StringBuilder

1) Difference between String and StringBuffer?

Case 1:

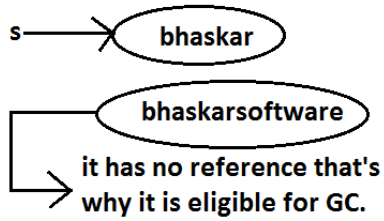
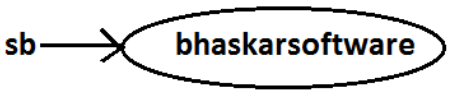
String s=new String("bhaskar");
s.concat("software");
System.out.println(s);//bhaskar
• Once we create a String object we can't perform any changes in the existing object. If we are try to perform any changes with those changes a new object will be created. This behavior is called immutability of the String object.
Diagram:
 
 	
	StringBuffer sb=new StringBuffer("bhaskar");
	sb.append("software");
	System.out.println(sb);
	• Once we created a StringBuffer object we can perform any changes in the existing object. This behavior is called mutability of the StringBuffer object.
	Diagram:
	 

Case 2:

String s1=new String("bhaskar");
String s2=new String("bhaskar");
System.out.println(s1==s2);//false
System.out.println(s1.equals(s2));//true
• In String class .equals() method is overridden for content compression hence if the content is same .equals() method returns true even though objects are different.
	StringBuffer sb1=new StringBuffer("bhaskar");
	StringBuffer sb2=new StringBuffer("bhaskar");
	System.out.println(sb1==sb2);//false
	System.out.println(sb1.equals(sb2));//false
	• In StringBuffer class .equals() method is not overridden for content compression hence Object class .equals() method got executed which is always meant for reference compression. Hence if objects are different .equals() method returns false even though content is same.

2) What is the meaning of immutability and mutability?

String s=new String("bhaskar");
s.concat("software");
System.out.println(s);//bhaskar
• Once we create a String object we can't perform any changes in the
	StringBuffer sb=new StringBuffer("bhaskar");
	sb.append("software");
	System.out.println(sb);
	• Once we created a StringBuffer object we can perform any changes in

existing object. If we are try to perform any changes with those changes a new object will be created. This behavior is called immutability of the String object. **Diagram:** 	the existing object. This behavior is called mutability of the StringBuffer object. **Diagram:** 

3) Why String objects are immutable where as StringBuffer objects are mutable?

- In the case of String as several references pointing to the same object, by using one reference if we are allowed perform the change the remaining references will be impacted. To prevent this once we created a String object we can't perform any change in the existing object that is immutability is only due to SCP.
- But in the case of StringBuffer for every requirement we are creating a separate object by using one reverence if we are performing any change in the object the remaining references won't be impacted hence immutability concept is not require for the StringBuffer.

4) With respect to .equals() method What is the difference between String and StringBuffer?

String s1=new String("bhaskar");
String s2=new String("bhaskar");
System.out.println(s1==s2);//false
System.out.println(s1.equals(s2));//true
• In String class .equals() method is overridden for content compression hence if the content is same .equals() method returns true even though objects are different.
	StringBuffer sb1=new StringBuffer("bhaskar");
	StringBuffer sb2=new StringBuffer("bhaskar");
	System.out.println(sb1==sb2);//false
	System.out.println(sb1.equals(sb2));//false
	• In StringBuffer class .equals() method is not overridden for content compression hence Object class .equals() method got executed which is always meant for reference compression. Hence if objects are different .equals() method returns false even though content is same.

5) What is String constant pool?

- A specially designed memory area for the String literals.

6) What are various advantages of string constant pool?

- Instead of creating a separate object for every requirement we can create only one object and we can reuse same object for every requirement. This approach improves performance and memory utilization.

7) Is garbage collector allowed to destroy objects present in SCP?

1) Garbage collector can't access SCP area hence even though object doesn't have any reference still that object is not eligible for GC if it is present in SCP.

8) When SCP objects will be destroyed?

- All SCP objects will be destroyed at the time of JVM shutdown automatically.

9) Is it possible to create our own immutable class? Give an example?

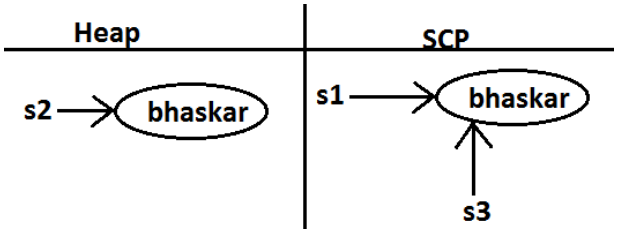
Creation of our own immutable class:

- We can create our own immutable classes.
- Once we created an object we can't perform any changes in the existing object. If we are trying to perform any changes with those changes a new object will be created. If there is no change in the content then existing object will be reused. This behavior is called immutability.

Example:

```
class StringInternDemo
{
        public static void main(String[] args)
        {
        String s1="bhaskar";
        String s2=s1.toUpperCase();
        String s3=s1.toLowerCase();
        System.out.println(s1==s2);//false
        System.out.println(s1==s3);//true
        }
}
```

Diagram:



10) Declaring a variable as final, it is possible to make that object is immutable?

Final vs immutability:

- If we declare a variable as final then we can't perform reassignment for that variable. It doesn't mean in the corresponding object we can't perform any changes. That is

through final keyword we won't get any immutability that is final and immutability concepts are different.

Example:

```
class Test
{
        public static void main(String[] args)
        {
                final StringBuffer sb=new StringBuffer("bhaskar");
                sb.append("software");
                System.out.println(sb);//bhaskarsoftware
                sb=new StringBuffer("solutions");//C.E: cannot assign a value to final
variable sb
        }
}
```

- In the above example even though "sb" is final we can perform any type of change in the corresponding object. That is through final keyword we are not getting any immutability nature.

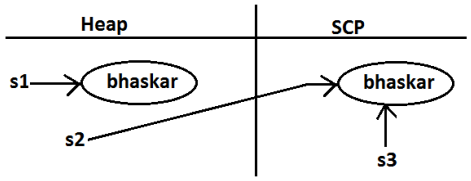
11) In java which classes are immutable?

- In addition to String all wrapper objects are immutable in java.

12) What is the interning of the String?

- By using heap object reference, if we want to corresponding SCP object reference we should go for intern() method.

Diagram:



13) What is trim?

- We can use this method to remove blank spaces present at beginning and end of the string but not blank spaces present at middle of the String.

14) What is the difference between StringBuffer and StringBuilder?

StringBuffer	StringBuilder
1) Every method present in StringBuffer is synchronized.	1) No method present in StringBuilder is synchronized.
2) At a time only one thread is allow to operate on the StringBuffer object hence StringBuffer object is Thread	2) Multiple Threads are allowed to operate simultaneously on the StringBuilder object hence

safe.	StringBuilder is not Thread safe.
3) It increases waiting time of the Thread and hence relatively performance is low.	3) Threads are not required to wait and hence relatively performance is high.
4) Introduced in 1.0 version.	4) Introduced in 1.5 versions.

15) What is StringBuilder?

- Every method present in StringBuffer is declared as synchronized hence at a time only one thread is allowed to operate on the StringBuffer object due to this, waiting time of the threads will be increased and effects performance of the system.
- To overcome this problem sun people introduced StringBuilder concept in 1.5v.

16) When we should go for String, StringBuffer and StringBuilder?

- If the content is fixed and won't change frequently then we should go for String.
- If the content will change frequently but Thread safety is required then we should go for StringBuffer.
- If the content will change frequently and Thread safety is not required then we should go for StringBuilder.

Wrapper Classes and AutoBoxing

1) Explain the purpose of wrapper classes?

The main objectives of wrapper classes are:

- To wrap primitives into object form so that we can handle primitives also just like objects.

2) What are various wrapper classes present in java?

Wrapper class	Constructor summery
Byte	byte, String
Short	short, String
Integer	Int, String
Long	long, String
Float	float, String, double
Double	double, String
Character	char,~~String~~
Boolean	boolean, String

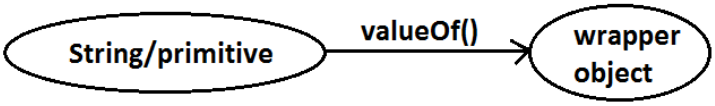
3) How to create a wrapper class object?

4) What is the purpose of valueOf()?

valueOf() method: We can use valueOf() method to create wrapper object for the given primitive or String this method is alternative to constructor.

public static wrapper valueOf(String s);

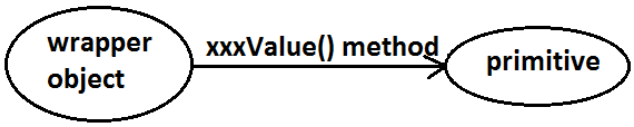
Diagram:



5) What is the purpose of xxxValue()?

xxxValue() method: We can use xxxValue() method to convert wrapper object to primitive.

Diagram:



6) How we can find primitive from wrapper object?

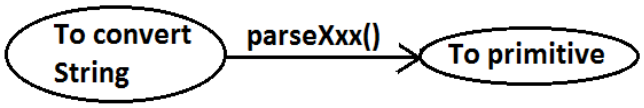
- By using xxxValue() method we can find primitive from wrapper object.

7) What is the purpose of parseXxx()?

parseXxx() method: We can use this method to convert String to corresponding primitive.

public static primitive parseXxx(String s);

Diagram:



8) How to convert primitive to string?

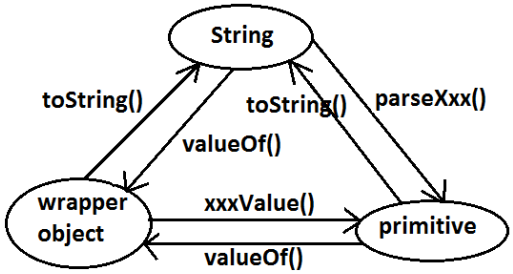
toString() method: We can use toString() method to convert wrapper object (or) primitive to String.

public StringtoString();

9) Explain the methods which are required to perform the following conversions:

1) **int--------→String**
2) **String----→int**
3) **int--------→Integer**
4) **Integer---→int**
5) **Integer---→String**
6) **String-----→Integer**

Diagram:



10) What is Auto Boxing & Auto Unboxing?

AutoBoxing: Automatic conversion of primitive to wrapper object by compiler is called Autoboxing.

Example:

Integer i=10; [compiler converts "int" to "Integer" automatically by AutoBoxing]

- After compilation the above line will become.

Integer i=Integer.valueOf(10);

- That is internally AutoBoxing concept is implemented by using valueOf() method.

Autounboxing: Automatic conversion of wrapper object to primitive by compiler is called Autounboxing.

Example:

Integer i=new Integer(10);

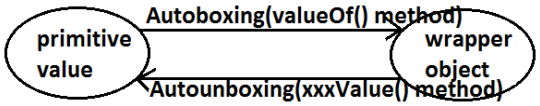
Int i=I; [compiler converts "Integer" to "int" automatically by Autounboxing]

- After compilation the above line will become.

Int i=I.intValue();

- That is Autounboxing concept is internally implemented by using xxxValue() method.

Diagram:



11) Explain the new features which are introduced in 1.5 versions?

1.5 versions new features

1) For-Each
2) Var-arg
3) Queue
4) Generics
5) Autoboxing and Autounboxing
6) Co-varient return types
7) Annotations
8) Enum
9) Static import
10) String builder

12) While performing overloading method resolution in which order compiler will give the precedence in following cases:

Widening: Converting a lower data type into a higher data type is called widening.

1) **Widening vs var.arg:**Widening dominates var-arg method.
2) **Var.arg vs AutoBoxing:**AutoBoxing dominates var-arg method.
3) **AutoBoxing vs widening:**Widening dominates AutoBoxing.

- In general var-arg method will get least priority. That is if no other method matched then only var-arg method will get chance. It is exactly same as “default” case inside a switch.

13) Which objects are immutable in java?

- In addition to String all wrapper objects also immutable in java.

14) Which wrapper classes are direct child classes of Object?

- Boolean and Character

15) Which wrapper classes are not child classes of Object?

- Byte, Short, Integer, Long, Float, Double.

16) Which wrapper classes are child classes of Number?

- Byte, Short, Integer, Long, Float, Double.

FileI/O

1) How to create a file and directory programmatically?

Requirement: Write code to create a directory named with bhaskar123 in current working directory and create a file named with abc.txt in that directory.

Program:

```
import java.io.*;
class  FileDemo
{
        public static void main(String[] args)throws IOException
        {
                File f1=new File("bhaskar123");
                f1.mkdir();
                File f2=new File("bhaskar123","abc.txt");
                f2.createNewFile();
        }
}
```

2) Can you explain the impact of the line?

File f=new File("bhaskar.txt");

- This line 1st checks whether abc.txt file is already available (or) not if it is already available then "f" simply refers that file. If it is not already available then it won't create any physical file just creates a java File object represents name of the file.

3) What is the difference between FileReader and BufferedReader?

FileReader: By using FileReader we can read character data from the file.

BufferedReader: This is the most enhanced(better) Reader to read character data from the file.

- BufferedReader cannot communicate directly with the File it should communicate via some Reader object. The main advantage of BufferedReader over FileReader is we can read data line by line instead of character by character.

4) Which is the most enhanced Reader?

- The most enhanced Reader to read character data from the File is BufferedReader.

5) What is the limitation of FileWriter?

- While writing data by FileWriter compulsory we should insert line separator(\n) manually which is a bigger headache to the programmer.
- To overcome these limitations we should go for BufferedWriter concept.

6) What is the difference between FileWriter and BufferedWritter?

FileWriter: By using FileWriter we can write character data to the file.

BufferedWriter: By using BufferedWriter object we can write character data to the file.

7) What is the difference between BufferedWritter and PrintWriter?

- By using FileWriter and BufferedWriter we can write only character data to the File but by using PrintWriter we can write any type of data to the File.

8) What is the most enhanced Writer?

- The most enhanced Writer to write character data to the File is PrintWriter.

9) Explain about flush() method?

- To give the guarantee the last character of the data also added to the file.

10) What is the difference between write(100) and print(100) of PrintWriter class?

- In the case of write(100) the corresponding character “d” will be added to the File but in the case of print(100) “100” value will be added directly to the File.

11) What is the difference between Readers-Writers and input output streams?

- In general we can use Readers and Writers to handle character data. Where as we can use InputStreams and OutputStreams to handle binary data(like images, audio files, video files etc).
- We can use OutputStream to write binary data to the File and we can use InputStream to read binary data from the File.