1. **Explain a virtual function?**

A virtual function is a member function that you expect to be redefined in derived classes. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function. Virtual functions ensure that the correct function is called for an object, regardless of the expression used to make the function call. Suppose a base class contains a function declared as virtual and a derived class defines the same function. The function from the derived class is invoked for objects of the derived class, even if it is called using a pointer or reference to the base class.

1. **Explain a friend function?**

 A friend function can be given a special grant to access private and protected members. A friend function can be:   
a) A member of another class   
b) A global function

By using the keyword friend compiler knows the given function is a friend function.

For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.

**Characteristics of a Friend function:**

* The function is not in the scope of the class to which it has been declared as a friend.
* It cannot be called using the object as it is not in the scope of that class.
* It can be invoked like a normal function without using the object.
* It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
* It can be declared either in the private or the public part.

JAVA doesn't support “friend” keyword but the concept of friend function can be implemented in JAVA by choosing appropriate access specifiers for the class and its members.

1. **Explain function overloading?**

Function overloading is a feature of object oriented programming where two or more functions can have the same name but different parameters.

When a function name is overloaded with different jobs it is called Function Overloading.

In Function Overloading “Function” name should be the same and the arguments should be different.

Function overloading can be considered as an example of polymorphism feature

How Function Overloading works?

Exact match: - (Function name and Parameter)

If a not exact match is found:–

               ->Char, Unsigned char, and short are promoted to an int.

               ->Float is promoted to double

If no match found:

               ->tries to find a match through the standard conversion.

ELSE ERROR .

1. **Explain a sub class, super class?**

**Definitions:** A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*).

The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).

Excepting Object, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of Object.

Classes can be derived from classes that are derived from classes that are derived from classes, and so on, and ultimately derived from the topmost class, Object. Such a class is said to be *descended* from all the classes in the inheritance chain stretching back to Object.

The idea of inheritance is simple but powerful: When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class. In doing this, you can reuse the fields and methods of the existing class without having to write (and debug!) them yourself.

A subclass inherits all the *members* (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

1. **Write in brief linking of base class, sub class and base object, sub object.**

Object relation of Superclass (parent) to Subclass (child) exists while child to parent object relation never exists.This means that reference of parent class could hold the child object while child reference could not hold the parent object.

\In case of overriding of non-static method the runtime object would evaluate that which method would be executed of subclass or of superclass. While execution of static method depends on the type of reference that object holds.

Other basic rule of inheritance is related to static and non static method overriding that static method in java could not be overridden while non-static method can be.However subclass can define static method of same static method signature as superclass have hut that would not be consider as overriding while known as hiding of static method of superclass

1. **Explain an abstract class?**

An *abstract class* is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

If a class includes abstract methods, then the class itself *must* be declared abstract, as in:

public abstract class GraphicObject {

// declare fields

// declare nonabstract methods

abstract void draw();

}

An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon).

1. **Explain operator overloading?**

Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type. Operator overloading is used to overload or redefines most of the operators available

## Rules for Operator Overloading

* Existing operators can only be overloaded, but the new operators cannot be overloaded.
* The overloaded operator contains at least one operand of the user-defined data type.
* We cannot use friend function to overload certain operators. However, the member function can be used to overload those operators.
* When unary operators are overloaded through a member function take no explicit arguments, but, if they are overloaded by a friend function, takes one argument.
* When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

1. **Define different types of arguments? (Call by value/Call by reference)**

An argument is a way for you to provide more information to a function. The function can then use that information as it runs, like a variable. Said differently, when you create a function, you can pass in data in the form of an argument, also called a parameter.

Functions can be invoked in two ways: Call by Value or Call by Reference. These two ways are generally differentiated by the type of values passed to them as parameters.

The parameters passed to function are called actual parameters whereas the parameters received by function are called formal parameters.

**Call by Value:** In this parameter passing method, values of actual parameters are copied to function’s formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of the caller.

In call by values we cannot alter the values of actual variables through function calls.

**Call by Reference:** Both the actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in actual parameters of the caller.  
In call by reference we can alter the values of variables through function calls.

**9.** **Explain the super keyword?**

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

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

## Usage of Java super Keyword

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

**10**. **Explain method overriding?**

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java.**

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

### Usage of Java Method Overriding

* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* Method overriding is used for runtime polymorphism

#### Rules for Java Method Overriding

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

A Java manipulator is your own code in Java that takes records from any number of pipeline components in Forge or, optionally, your source data, and changes it according to your processing requirements.

2)Can you give some examples of tokens?

Java token includes the following:

* Keywords
* Identifiers
* Literals
* Operators
* Separators
* Comment

3) Explain structured programming and its disadvantage?

In structured programming design, programs are broken into different functions these functions are also known as **modules**, **subprogram**, **subroutines** and **procedures**.

Each function is design to do a specific task with its own data and logic. Information can be passed from one function to another function through parameters. A function can have local data that cannot be accessed outside the function’s scope. The result of this process is that all the other different functions are synthesized in an another function. This function is known as main function. Many of the high level languages supported structure programming.

Advantages

1. It is user friendly and easy to understand.
2. Similar to English vocabulary of words and symbols.
3. It is easier to learn.
4. They require less time to write.
5. They are easier to maintain.
6. These are mainly problem oriented rather than machine based.

Disadvantages

1. A high level language has to be translated into the machine language by translator and thus a price in computer time is paid.
2. The object code generated by a translator might be inefficient compared to an equivalent assembly language program.
3. Data type are proceeds in many functions in a structured program. When changes occur in those data types, the corresponding change must be made to every location that acts on those data types within the program. This is really a very time consuming task if the program is very large

4) Explain the advantage of C++ being a block-structured language?

Easier to read and understand

User Friendly

Easier to Maintain

Mainly problem based instead of being machine based

Development is easier as it requires less effort and time

Easier to Debug

Machine-Independent, mostly.

5) Can Struct be inherited?

**Structs don't provide inheritance**. It is not possible to inherit from a struct and a struct can't derive from any class. Similar to other types in

6) When to use interface over abstract class.

* If you are creating functionality that will be useful across a wide range of objects, then you must use an interface. Abstract classes, at the end of the day, should be used for objects that are closely related. But the interfaces are best suited for providing common functionality to unrelated cases.
* Interfaces are a great choice if you think that the API won’t be changing for a while.
* Interfaces are also a great choice. If you want to have something similar to the multiple inheritances, then you can implement various interfaces.
* If we are going to design the small, concise bits of functionality, then you must use interfaces. But if you are designing the large functional units, then you must use an abstract class.

But, at the end of the day, it all comes down to what you require and what you are going to do about it. The same functionalities can be accomplished with both interface and abstract classes. With the coding standards, the interface would help you accomplish the abstraction and polymorphism which are among the main principles of coding or programming.

It also helps to keep your code loosely coupled instead of tightly coupled, which happens when the high-level modules depend on the low-level modules as well. Interfaces can also be used for the dependency injection that makes it easier to mock the derived classes when testing.

If you want to become a coding expert or programming enthusiast, then join an [IT boot camp](https://www.quickstart.com/bootcamp/web-development/) right now to work on your skillset and acquire more knowledge on the subject. QuickStart’s web developer Bootcamp can help you earn the certifications you need to become an expert at web development.

7)Explain a private constructor? Where will you use it?

A private constructor in Java is used in restricting object creation. It is a special instance constructor used in static member-only classes. If a constructor is declared as private, then its objects are only accessible from within the declared class. You cannot access its objects from outside the constructor class.

* You can use it with static members-only classes.
* You can use it with [static utility](https://www.upgrad.com/blog/java-static-method-static-keyword/) or constant classes.
* You can use it to serve singleton classes.
* You can use it to assign a name, for instance, creation by utilising factory methods.

You can use it to prevent subclassing.

8) Can you override private virtual methods?

You can't declare private virtual methods since there'd be no way to override them But you can override protected virtual methods.

9)Can you allow class to be inherited, but prevent from being over-ridden?

You can prevent a class from being subclassed by using the final keyword in the class's declaration. Similarly, you can prevent a method from being overridden by subclasses by declaring it as a final method.

10)Why can’t you specify accessibility modifiers for methods inside interface?

Interface members are always public because the purpose of an interface is to enable other types to access a class or struct. No access modifiers can be applied to interface members. All the interface methods are

**1.Explain OOPS?**

OOPs, or Object-Oriented Programming is a programming model or paradigm which revolves around the concept of “**OBJECTS**”. Objects can be considered as real-world instances of entities like class, that contain some characteristics and behaviors specified in the class template.

In simple language, a class can be considered as the blueprint or template, based on which objects can be created. So the Objects are considered the instance of a class, and are therefore sometimes called “**instances**”. The term “**characteristics**” refers to the “what” about the Object, and the term “**behavior**” refers to the “how” about the Object.

For example, if we consider a car, then based on the OOPs model:

Class = A specific car model, such as Audi A4, BMW I8, Maruti Suzuki Vitara Brezza, etc.

Object = A specific car of any model, like the car you own

Characteristics = What is the color of your car? What is the Chassis number of your car? etc

Behavior = How to start the car? How to change the gear of the car? etc.

Characteristics are also known as data, attributes, or properties, and Behaviours are also known as the functions, procedures or methods, in the programming language.

The concept of “objects” allows the OOPs model to easily access, use and modify the instance data and methods, interact with other objects, and define methods in runtime (during the execution of the program). This gives the OOPs model significance and makes it diverse in its implementation.

**2.Explain an abstraction? Real life example.**

Abstraction is a feature of OOPs. The feature allows us to **hide** the implementation detail from the user and shows only the functionality of the programming to the user. Because the user is not interested to know the implementation. It is also safe from the security point of view.

Let's understand the abstraction with the help of a real-world example. The best example of abstraction is a car. When we derive a car, we do not know **how is the car moving** or **how internal components are working?** But we know **how to derive a car**. It means it is not necessary to know how the car is working, but it is important how to derive a car. The same is an abstraction.

* We can achieve abstraction in two ways:

1. Using Abstract Class
2. Using Interface

**3.Explain encapsulation? Real life example.**

**Encapsulation in Java** is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.



We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The **Java Bean** class is the example of a fully encapsulated class.

**4.Explain the relationship among abstraction and encapsulation?**

They both seem very similar but totally different in concept and implementation. The **major difference between abstraction and encapsulation** is that abstraction hides the code complexity while encapsulation hides the internal working from the outside world.

Difference between Abstraction and Encapsulation:

|  | ABSTRACTION | ENCAPSULATION |
| --- | --- | --- |
| 1. | Abstraction is the process or method of gaining the information. | While encapsulation is the process or method to contain the information. |
| 2. | In abstraction, problems are solved at the design or interface level. | While in encapsulation, problems are solved at the implementation level. |
|  |  |  |
| 3. | Abstraction is the method of hiding the unwanted information. | Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside. |
| 4. | We can implement abstraction using abstract class and interfaces. | Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public. |
| 5. | In abstraction, implementation complexities are hidden using abstract classes and interfaces. | While in encapsulation, the data is hidden using methods of getters and setters. |
| 6. | The objects that help to perform abstraction are encapsulated. | Whereas the objects that result in encapsulation need not be abstracted. |

**5.Explain polymorphism?**

**Polymorphism** is an object-oriented programming concept that refers to the ability of a *variable*, *function* or *object* to take on *multiple* forms. In a programming language exhibiting **polymorphism**, class objects belonging to the same hierarchical tree (inherited from a common **parent** *class*) may have functions with the same name, but with different behaviors.

**Example**

The classic example is of the **Shape** *class* and all the *classes* that are *inherited* from it, such as: *Rectangle, Triangle*, *Circle.*

**6.Explain Inheritance?**

**Inheritance** is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance facilitates Reusability and is an important concept of OOPs.

**Java Inheritance** is a mechanism in which one class acquires the property of another class. In Java, when an “Is-A” relationship exists between two classes, we use Inheritance. The parent class is called a super class and the inherited class is called a subclass. The keyword extends is used by the sub class to inherit the features of super class.

## Types of Inheritance

Here are the different types of inheritance in Java:

### Single Inheritance:

In Single Inheritance one class extends another class (one class only).

### Multilevel Inheritance:

In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.

### Hierarchical Inheritance:

In Hierarchical Inheritance, one class is inherited by many sub classes.

### Hybrid Inheritance:

Hybrid inheritance is one of the inheritance types in Java which is a combination of Single and Multiple inheritance.

Java doesn’t support hybrid/Multiple inheritance.

**7.How composition is better than inheritance?**

**Composition offers better test-ability of a class than Inheritance**. If one class consists of another class, you can easily construct a Mock Object representing a composed class for the sake of testing. This privilege is not given by inheritance.

Whereas inheritance derives one class from another, composition defines a class as the sum of its parts. Classes and objects created through inheritance are **tightly** coupled because changing the parent or superclass in an inheritance relationship risks breaking your code.

**8.Which OOPS concept is used as a reuse mechanism?**

Reusability: Inheritance supports the concept of “reusability”, i.e. when we want to create a **new class** and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

**9.Which OOPS concept exposes only the necessary information to the calling functions?**

Data hiding is a **software development technique specifically used in** object-oriented programming (OOP) to hide internal object details (data members). Data hiding ensures exclusive data access to class members and protects object integrity by preventing unintended or intended changes.

**10.Explain a class? Create a class.**

Definition: A class is a blueprint that defines the variables and the methods common to all objects of a certain kind.

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

* **Fields**
* **Methods**
* **Constructors**
* **Blocks**
* **Nested class and interface**

To create a class, use the keyword class:

### **Main.java**

Create a class named "Main" with a variable x:

public class Main {

int x = 5;

}

Q.11 Using above created class, Write in brief abstraction and encapsulation.

--> abstract class AbstractClass {

public abstract String getName();

public abstract void setName(String name);

}

class A extends AbstractClass {

private String name;

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

class Demo {

public static void main(String args[]) {

A obj = new A();

obj.setName("Your name");

System.out.println(obj.getName());

}

}

Abstraction :- Hiding the complexity and showing only essential details is called as Abstraction.

In above code,there is class named AbstractClass.In that class,getName() and setName(String) these are the methods where we unable to see there the implementation of these methods means we are hiding the complexities and we know only the method names which we can use which shows us that what methods will do(functionality),so we have only the essential details.

Encapsulation :- It is a process of wrapping code and data together into a single unit.

From above example,In class A there is one String type variable(data) and two methods(code) which are using that variable are bind together into a single class A.Hence,there is Encapsulation.

Q.12 Explain difference among class and object?

--> Class- 1. Class is used as a template for declaring and creating the objects.

Object- 1. An object is an instance of a class.

Class- 2. When a class is created, no memory is allocated.

Object- 2. Objects are allocated memory space whenever they are created.

Class- 3. The class has to be declared only once.

Object- 3. An object is created many times as per requirement.

Class- 4. A class cannot be manipulated as they are not available in the memory.

Object- 4. Objects can be manipulated.

Class- 5. A class is a logical entity.

Object- 5. An object is a physical entity.

Class- 6. It is declared with the class keyword.

Object- 6. It is created with the new keywords in Java.

Class- 7. Class does not contain any values which can be associated with the field.

Object- 7. Objects are like a variable of the class.

Q.13 Define access modifiers?

--> The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers-

Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Q.14 Explain an object? Create an object of above class.

--> An entity that has state and behavior is known as an object.For e.g. chair,bike,pen,table,car,etc.These are real world objects.But object can be physical or logical.

Object has three characteristics-

State: represents the value of an object.

Behavior: represents the behavior of an object.

Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However,it is used internally by the JVM to identify each object uniquely.

Another definition of Object is,It is an instance of a class. A class is a blueprint from which objects are created. So,an object is the instance of a class.

For example-

class Example {

int a = 5;

void display(){

System.out.println("Value of a = "+a);

}

}

class Demo {

public static void main(String args[]) {

Example obj = new Example();

}

}

Q.15 Give real life examples of object.

--> Ex. 1. Planet is an object which has states like its diameter,size,shape,color,human existence,water existence,temperature,rotation speed,gravity,etc and behaviours like rotation around sun,rotation around itself,changing temperature,changing environment,etc.

Ex. 2. Bike is an object which has states like its company name,model name,color,price,maximum speed,mileage,etc and behaviours like start bike,stop bike,consume fuel,driving,accelerate speed,changing gears,brake,etc.

Q.16 Explain a Constructor.

--> A constructor is a block of codes similar to the method. It is called when an instance of the class

is created. At the time of calling constructor, memory for the object is allocated in the memory.It is a special type of method which is used to initialize the object.Every time an object is created using the new() keyword, at least one constructor is called.It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java-

1. Default(No-argument) constructor - A constructor is called Default constructor when it doesn't have any parameter.

2.Parameterized constructor - A constructor which has a specific number of parameters is called a Parameterized constructor.

There are certain rules defined for the constructor-

1. Constructor name must be the same as its class name.

2. A Constructor does not have any return type.

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

Q.17 Define the various types of constructors?

--> There are two types of constructors in Java-

1. Default(No-argument) constructor - A constructor is called Default constructor when it doesn't have any parameter.

Syntax:- <ClassName>(){

}

2.Parameterized constructor - A constructor which has a specific number of parameters is called a Parameterized constructor.

Syntax:- <ClassName>(parameters list){

}

Q.18 Whether static method can use nonstatic members?

--> NO,Static method can not use non-static members.Static method gives direct access only to the static members.If we trying to access non-static members then compiler gives us the error like non-static members can not have access inside static context.But we can use non-static members inside static method by creating object of the particular class of non-static members in static method and can access using reference variable and dot operator.

Q.19 Explain Destructor?

--> Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed.A destructor is the last function that is going to be called before an object is destroyed. If the object is created by using new or the constructor uses new to allocate memory which resides in the heap memory or the free store, the destructor should use delete to free the memory.

Properties of Destructor-

1. Destructors don’t take any argument and don’t return anything.

2. Destructors have same name as the class preceded by a tilde (~).

3. It cannot be declared static or const.

4. The destructor does not have arguments.

5. It has no return type not even void.

6. An object of a class with a Destructor cannot become a member of the union.

7. A destructor should be declared in the public section of the class.

8. The programmer cannot access the address of destructor.

In Java,there is no concept of destructor because the work of destructor is automatically done by garbage collector.This is the one of the feature of Java.

Q.20 Explain an Inline function?

--> Inline function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality.

To inline a function, place the keyword inline before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line.

A function definition in a class definition is an inline function definition, even without the use of the inline specifier.

Java does not provide inline functions it is typically done by the JVM at execution time.

41. Do we require a parameter for constructors?

There are two types of constructors defined in java namely default constructor, Parametrized constructor. There is a minute difference between default constructor and Parametrized constructor. The default constructor is a type of constructor which has no arguments but yes object instantiation is performed there also.

On the other hand, as the name suggests Parametrized constructor is a special type of constructor where an object is created, and further parameters are passed to distinct objects.

With a parameterized constructor for a class, one must provide initial values as arguments, otherwise, the compiler reports an error.

42. Explain static and dynamic binding?

1. Static Binding (also known as Early Binding) :

When type of the object is determined at compiled time(by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

- Example of static binding

class Dog

{

private void eat(){System.out.println("dog is eating...");}

public static void main(String args[])

{

Dog d1=new Dog();

d1.eat();

}

}

2. Dynamic Binding (also known as Late Binding) :

When type of the object is determined at run-time, it is known as dynamic binding.

- Example of dynamic binding

class Animal

{

void eat()

{

System.out.println("animal is eating...");

}

}

class Dog extends Animal

{

void eat()

{

System.out.println("dog is eating...");

}

public static void main(String args[])

{

Animal a=new Dog();

a.eat();

}

}

Output : dog is eating...

In the above example object type cannot be determined by the compiler, because the instance of Dog is also an instance of Animal.

So compiler doesn't know its type, only its base type.

43. How many instances can be created for an abstract class?

No, you cannot create an instance of an abstract class because it does not have a complete implementation. The purpose of an abstract class

is to function as a base for subclasses. It acts like a template, or an empty or partially empty structure, you should extend it and build on it before you can use it.

44. Explain the default access specifiers in a class definition?

Default : When no access modifier is specified for a class, method, or data member – It is said to be having the default access modifier by default.

The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible only within the same package.

It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

- Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public,

so it cannot be accessed from outside the package.

//save by A.java

package pack;

class A

{

void msg()

{

System.out.println("Hello");

}

}

//save by B.java

package mypack;

import pack.\*;

class B

{

public static void main(String args[])

{

A obj = new A();//Compile Time Error

obj.msg();//Compile Time Error

}

}

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

45. Which OOPS concept is used as reuse mechanism?

Inheritance is the OOPS concept that can be used as reuse mechanism.

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.The idea behind inheritance in Java is

that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class.

Moreover, you can add new methods and fields in your current class also.

46. Define the Benefits of Object Oriented Programming?

Object Oriented Programming are

1. Re-usability

When we say re-usability, it means that “write once, use it multiple times” i.e., reusing some facilities rather than building it again and again, which can be achieved by using class. We can use it n number of times whenever required.

2. Data redundancy

It is one of the greatest advantages in oops. This is the condition which is created at the data storage when the same piece of data is held at two different places. If we want to use a similar functionality in multiple classes, we can just write common class definitions for the similar functionalities by inheriting them.

3. Code maintenance

It is easy to modify or maintain existing code as new objects which can be created with small differences for the existing ones. It also helps users from doing rework many times. It is time saving as we modify the existing codes incorporating new changes to it.

4. Security

Data hiding and abstraction are used to filter out limited exposure which means we are providing only necessary data to view as we maintain security.

5. Design benefits

The designers will have a longer and extensive design phase, which results in better designs. At a point of time when the program has reached critical limits, it will be easier to program all non oops one separately.

6. Easy troubleshooting

Using encapsulation objects are self-constrained. So, if developers face any problem easily it can be solved. And there will be no possibility of code duplicity.

47. Explain method overloading?

- Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters

or type of input parameters or both.

- Overloading is related to compile-time (or static) polymorphism.

- Method overloading increases the readability of the program.

48. Explain the difference among early binding and late binding?

Early Binding Late Binding

1. It is a compile-time process 1. It is a run-time process

2. The method definition and method call are 2. The method definition and method call are linked during the run time.

linked during the compile time.

3. Actual object is not used for binding. 3. Actual object is used for binding.

4. For example: Method overloading 4. For example: Method overriding

5. Program execution is faster 5. Program execution is slower

49. Explain early binding? Give examples?

Early Binding :

When type of the object is determined at compiled time(by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

- Example of static binding

class Dog

{

private void eat(){System.out.println("dog is eating...");}

public static void main(String args[])

{

Dog d1=new Dog();

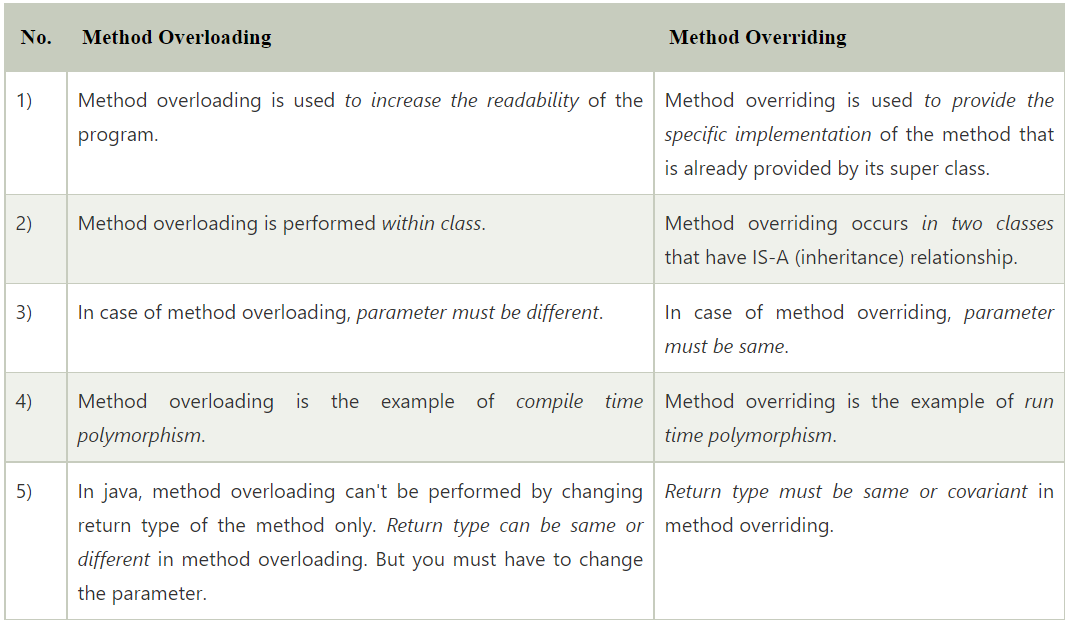
d1.eat();

}

}

**Q31. Difference among overloading and overriding?**

**Answer=>**



**Q32. Whether static method can use non-static members?**

**Answer=>**

Yes, a static method can access a non-static variable. This is done by creating an object to the class and accessing the variable through the object.

**Q33. Explain a base class, sub class, super class?**

**Answer=>**

The class from where a subclass inherits the features is called**superclass**. It is also called a base class or parent class.

A class that inherits all the members (fields, method, and nested classes) from the other class is called **subclass**. It is also called a derived class, child class, or extended class.

A base class is a class, in an object-oriented programming language, from which other classes are derived. It facilitates the creation of other classes that can reuse the code implicitly inherited from the base class (except constructors and destructors). A programmer can extend base class functionality by adding or overriding members relevant to the derived class.  
  
A base class may also be called parent class or superclass.

**Q34**. **Write in brief linking of base class, sub class and base object, sub object.**

**Answer=>**

Object relation of Superclass (parent) to Subclass (child) exists while child to parent object relation never exists.This means that reference of parent class could hold the child object while child reference could not hold the parent object.\In case of overriding of non-static method the runtime object would evaluate that which method would be executed of subclass or of superclass. While execution of static method depends on the type of reference that object holds.Other basic rule of inheritance is related to static and non static method overriding that static method in java could not be overridden while non-static method can be.However subclass can define static method of same static method signature as superclass have hut that would not be consider as overriding while known as hiding of static method of superclass**.**

**Q35. Explain an interface?**

Answer=> An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also **represents the IS-A relationship**.

It cannot be instantiated just like the abstract class.

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

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

Syntax:

1. **interface** <interface\_name>{
3. // declare constant fields
4. // declare methods that abstract
5. // by default.
6. }

**Q36. Explain exception handling?**

**Answer=>**

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

In this tutorial, we will learn about Java exceptions, it's types, and the difference between checked and unchecked exceptions. In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

**Q37. Explain the difference among structure and a class?**

**Answer=>** Technically speaking, structs and classes are almost equivalent, still there are many differences. The major difference like class provides the flexibility of combining data and methods (functions ) and it provides the re-usability called inheritance. Struct should typically be used for grouping data. The technical difference comes down to subtle issues about default visibility of members. Here you can see some of the Difference between Class and Structure.

Class is pass-by-reference and Struct is pass-by-copy, it means that, Class is a reference type and its object is created on the heap memory where as structure is a value type and its object is created on the stack memory. More about ....  Class can create a subclass that will inherit parent's properties and methods, whereas Structure does not support the inheritance.

A class has all members private by default. A struct is a class where members are public by default.

Classes allow to perform cleanup (garbage collector) before object is deallocated because garbage collector works on heap memory. Objects are usually deallocated when instance is no longer referenced by other code. Structures can not be garbage collector so no efficient memory management.

Classes are still fit for larger or complex objects and Structs are good for small, isolated model objects. Boxing and unboxing operations are used to convert between a struct type and object. Too much boxing and unboxing can have a negative impact on the heap, the garbage collector, and ultimately the performance of the application.

In general, it's perfectly possible to create structs that look a lot like classes and classes that look a lot like structs.

Which style you use depends on circumstances and taste. I usually prefer to use struct for classes that have all data public. I think of such classes as "not quite proper types, just data structures."

Q38. **Explain the default access modifier in a class?**

**Answer=>**

If a class member doesn’t have any access modifier specified, then it’s treated with default access. The access rules are similar to classes and the class member with default access will be accessible to the classes in the same package only. This access is more restricted than public and protected but less restricted than private.

(Least Accessible) **private** < **default** < **protected** < **public** (Most Accessible)

The below table summarise above access modifiers with respect to different classes in the same package or other packages and subclasses.

**Q39. Explain a pure virtual function?**

**Answer=>**

A virtual function or virtual method in an OOP language is a function or method used to override the behavior of the function in an inherited class with the same signature to achieve the polymorphism.

When the programmers switch the technology from [C++](https://www.javatpoint.com/cpp-tutorial) to [Java](https://www.javatpoint.com/java-tutorial), they think about where is the virtual function in Java. In C++, the virtual function is defined using the **virtual** keyword, but in Java, it is achieved using different techniques. See [Virtual function in C++](https://www.javatpoint.com/cpp-virtual-function).

Java is an object-oriented programming language; it supports OOPs features such as [polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java), abstraction, [inheritance](https://www.javatpoint.com/inheritance-in-java), etc. These concepts are based on objects, classes, and member functions.

By default, all the instance methods in Java are considered as the Virtual function except final, static, and private methods as these methods can be used to achieve polymorphism.

How to use Virtual function in Java

The virtual keyword is not used in Java to define the virtual function; instead, the virtual functions and methods are achieved using the following techniques:

* We can override the virtual function with the inheriting class function using the same function name. Generally, the virtual function is defined in the parent class and override it in the inherited class.
* The virtual function is supposed to be defined in the derived class. We can call it by referring to the derived class's object using the reference or pointer of the base class.
* A virtual function should have the same name and parameters in the base and derived class.
* For the virtual function, an IS-A relationship is necessary, which is used to define the class hierarchy in inheritance.
* The Virtual function cannot be private, as the private functions cannot be overridden.
* A virtual function or method also cannot be final, as the final methods also cannot be overridden.
* Static functions are also cannot be overridden; so, a virtual function should not be static.
* By default, Every non-static method in Java is a virtual function.
* The virtual functions can be used to achieve oops concepts like runtime polymorphism.

**Q40. Explain dynamic or run time polymorphism?**

**Answer=>**

It is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime, thats why it is called runtime polymorphism. I have already discussed method overriding in detail in a separate tutorial, refer it: [Method Overriding in Java](https://beginnersbook.com/2014/01/method-overriding-in-java-with-example/).

**Example**  
In this example we have two classes ABC and XYZ. ABC is a parent class and XYZ is a child class. The child class is overriding the method myMethod() of parent class. In this example we have child class object assigned to the parent class reference so in order to determine which method would be called, the type of the object would be determined at run-time. It is the type of object that determines which version of the method would be called (not the type of reference).

To understand the concept of overriding, you should have the basic knowledge of [inheritance in Java](https://beginnersbook.com/2013/03/inheritance-in-java/).

class ABC{

public void myMethod(){

System.out.println("Overridden Method");

}

}

public class XYZ extends ABC{

public void myMethod(){

System.out.println("Overriding Method");

}

public static void main(String args[]){

ABC obj = new XYZ();

obj.myMethod();

}

}

**Output:**

Overriding Method

50. Explain loose coupling and tight coupling?

1. Loose Coupling

- Loose coupling is a design goal to reduce the inter-dependencies between components of a system with the goal of reducing the risk that changes in one component

will require changes in any other component.

- Loose coupling is a much more generic concept intended to increase the flexibility of the system, make it more maintainable and makes the entire framework more stable.

2. Tight Coupling

- Tight coupling means classes and objects are dependent on one another. In general, tight coupling is usually not good because it reduces the flexibility and re-usability

of the code while Loose coupling means reducing the dependencies of a class that uses the different class directly.

- The tightly coupled object is an object that needs to know about other objects and is usually highly dependent on each other's interfaces.

- Changing one object in a tightly coupled application often requires changes to a number of other objects.

- In the small applications, we can easily identify the changes and there is less chance to miss anything. But in large applications, these inter-dependencies are not

always known by every programmer and there is a chance of overlooking changes.

1. How will you relate unrelated classes or how will you achieve polymorphism without using base class?
2. Explain Diamond problem?

a copy of both methods should be created in the subclass object which leaves the subclass with two methods with same prototype (name and arguments). Then, if you call the demo() method using the object of the subclass compiler faces an ambiguous situation not knowing which method to call. This issue is known as diamond problem in Java.

1. Explain the solution for diamond problem?

From Java8 on wards**default methods** are introduced in an interface. Unlike other abstract methods these are the methods of an interface with a default implementation. If you have default method in an interface, it is not mandatory to override (provide body) it in the classes that are already implementing this interface.

You can have same default methods (same name and signature) in two different interfaces and, from a class you can implement these two interfaces.

If you do so, you must override the default method from the class explicitly specifying the default method along with its interface name.

1. Explain the need of abstract class?

An abstract class **allows you to create functionality that subclasses can implement or override**. An interface only allows you to define functionality, not implement it. And whereas a class can extend only one abstract class, it can take advantage of multiple interfaces

1. Why can’t we instantiate abstract class?

We can't instantiate an abstract class because **the motive of abstract class is to provide a common definition of base class that multiple derived classes can share**

1. Can abstract class have constructors?

In abstract class, we have an instance variable, abstract methods, and non-abstract methods. We need to initialize the non-abstract methods and instance variables, therefore **abstract classes have a constructor**. ... This is also one of the reasons abstract class can have a constructor.

1. How many instances can be created for an abstract class?

**No you can't**, instead you can create instance of all other classes extending that abstract class. Because it's abstract and an object is concrete. An abstract class is sort of like a template, or an empty/partially empty structure, you have to extend it and build on it before you can use it.

1. Which keyword can be used for overloading?

If both parent & child classes have the same method, then the child class would override the method available in its parent class. By using the super keyword we can take advantage of both classes (child and parent) to achieve this.

We create an object of child class as it can inherit the parent class methods.

In the child class method, we call the method available in its parent class by using super().

1. Explain the default access specifiers in a class definition?

When no access modifier is specified for a class, method, or data member – It is said to be having the **default** access modifier by default.

* The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible **only within the same package**.

1. Define all the operators that cannot be overloaded?

operators can be overloaded. For an operator to be overloaded, at least one of the operands must be a user-defined object. Only existing operators can be overloaded. You cannot overload new operators

1. Explain the difference among structure and a class?

|  |  |
| --- | --- |
| Class | Structure |
| Classes are of reference types. | Structs are of value types. |
| All the reference types are allocated on heap memory. | All the value types are allocated on stack memory. |
| Allocation of large reference type is cheaper than allocation of large value type. | Allocation and de-allocation is cheaper in value type as compare to reference type. |
| Class has limitless features. | Struct has limited features. |
| Class is generally used in large programs. | Struct are used in small programs. |
| Classes can contain constructor or destructor. | Structure does not contain parameter less constructor or destructor, but can contain Parameterized constructor or static constructor. |
| Classes used new keyword for creating instances. | Struct can create an instance, with or without new keyword. |
| A Class can inherit from another class. | A Struct is not allowed to inherit from another struct or class. |
| The data member of a class can be protected. | The data member of struct can’t be protected. |
| Function member of the class can be virtual or abstract. | Function member of the struct cannot be virtual or abstract. |
| Two variable of class can contain the reference of the same object and any operation on one variable can affect another variable. | Each variable in struct contains its own copy of data(except in ref and out parameter variable) and any operation on one variable can not effect another variable. |

1. Explain the default access modifier in a class?

* The access level of a default modifier is only within the package.
* It cannot be accessed from outside the package.
* If we do not specify any access level or modifier, it is treated as default by default.
* It provides more accessibility than private. But, it is more restrictive than protected, and public.

.

73. Can you list out the different types of constructors?

76. Do We Require Parameter For Constructors?

80. Define The Various Types Of Constructors?

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

* Java Default Constructor

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

Syntax of default constructor:

<class\_name>(){}

* Java Parameterized Constructor

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

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

public class Main {

int x;

public Main(){ // This is Default Constructor

System.out.println(“This is Default Constructor”);

}

public Main(int y) { // This is Default Constructor

x = y;

}

public static void main(String[] args) {

Main myObj = new Main(5);

System.out.println(myObj.x);

}

}

74. Explain a friend function?

* A friend function of a class is defined outside that class's scope but it has the right to access all private and protected members of the class.
* Even though the prototypes for friend functions appear in the class definition, friends are not member functions.
* A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.
* To declare a function as a friend of a class, precede the function prototype in the class definition with keyword friend as follows −

class Box {

double width;

public:

double length;

friend void printWidth( Box box );

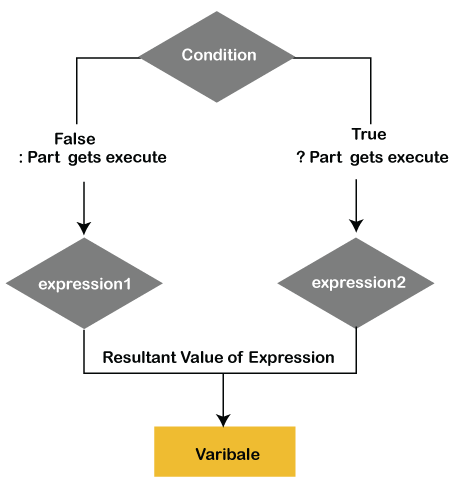
void setWidth( double wid );

};

75. Explain a ternary operator?

* In Java, the ternary operator is a type of Java conditional operator.
* The meaning of ternary is composed of three parts.
* The ternary operator (? :) consists of three operands.
* It is used to evaluate Boolean expressions.
* The operator decides which value will be assigned to the variable.
* It is the only conditional operator that accepts three operands.
* It can be used instead of the if-else statement.
* It makes the code much more easy, readable, and shorter.
* Every code using an if-else statement cannot be replaced with a ternary operator.
* Syntax:
* variable = (condition) ? expression1 : expression2
* The above statement states that if the condition returns true, expression1 gets executed, else the expression2 gets executed and the final result stored in a variable.

Ternary Operator Java



Example of Ternary Operator

TernaryOperatorExample.java

public class TernaryOperatorExample

{

public static void main(String args[])

{

int x, y;

x = 20;

y = (x == 1) ? 61: 90;

System.out.println("Value of y is: " + y);

y = (x == 20) ? 61: 90;

System.out.println("Value of y is: " + y);

}

}

Output:

Value of y is: 90

Value of y is: 61

77. Explain Sealed Modifiers?

* A sealed class is a class or interface which restricts which other classes or interfaces may extend it.
* Sealed classes and interfaces represent restricted class hierarchies that provide more control over an inheritance.
* All direct subclasses of a sealed class are known at compile time. No other subclasses may appear after a module with the sealed class is compiled.
* For example, third-party clients can't extend your sealed class in their code.
* Thus, each instance of a sealed class has a type from a limited set that is known when this class is compiled.

78. Explain The Difference Among New And Override?

* new means respect your REFERENCE type(left-hand side of =) , thereby running reference types's method. If redefined method doesn't have new keyword, it is behaved as it has. Moreover, it also known as non-polymorphic inheritance. That is, “I’m making a brand new method in the derived class that has absolutely nothing to do with any methods by the same name in the base class.” - by said Whitaker
* override, which must be used with virtual keyword in its base class, means respect your OBJECT type(right-hand side of =), thereby running method overriden in regardless of reference type. Moreover, it also known as polymorphic inheritance.

79. How Can We Call The Base Method Without Creating An Instance?

1. Its possible If its a static method.

Static methods are the methods in Java that can be called without creating an object of class. They are referenced by the class name itself or reference to the Object of that class.

2. By inheriting from that class.

3. Its possible from derived classes using base keyword.

91. Can static members use non static members? Give reasons. Ans.--> 1) A static method is a method that belongs to a class, but it does not belong to an instance of that class and this method can be called without the instance or object of tha t class. 2) Every methods in java defaults to non-static method without static keyword preceding it. non-static methods can access any static method and static variable also, without using the object of the class. 3) Static method uses compile time or early binding. 4) Non-static method can be overridden because of runtime binding. =================================================================================== ================================================== 92. Define different ways a method can be overloaded? Ans.--> 1) Method overloading can be achieved by the following: By changing the number of parameters in a method. By changing the order of parameters in a method. By using different data types for parameters. =================================================================================== =================================================== 93. Can we have an abstract class without having any abstract method? Ans.--> 1) Yes we can have an abstract class without Abstract Methods as both are independent concepts. Declaring a class abstract means that it can not be instantiated on its own and can only be sub classed. Declaring a method abstract means that Method will be defined in the subclass. =================================================================================== =================================================== 94. Explain the default access modifier of a class? Ans.--> Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc. A variable or method declared without any access control modifier is available to any other class in the same packa ge. =================================================================================== =================================================== 95. Can function overriding be explained in same class? Ans.--> While overriding − 1)Both methods should be in two different classes and, these classes must be in an inheritance relation. 2)Both methods must have the same name, same parameters and, same return type else they both will be treated as d ifferent methods. 3)The method in the child class must not have higher access restrictions than the one in the superclass. If you try to do so it raises a compile-time exception. 4)If the super-class method throws certain exceptions, the method in the sub-class should throw the same exception or its subtype (can leave without throwing any excepti on). Therefore, you cannot override two methods that exist in the same class, you can just overload them. =================================================================================== ======================================================== 96. Does function overloading depends on Return Type? Ans.--> No,It does not depend on Return Type. Because if return type is different and function name as well as para meter is also same The return type of a function does not create any effect on function overloading. Same function signature with different return type will not be overloaded. =================================================================================== ========================================================= 97. Define different ways to declare an array? Ans.-->1) A Java array variable can also be declared like other variables with [] after the data type. The variables in the array are ordered and each have an index beginning from 0. Java array can be also be used as a static field, a local variable or a method parameter. 2) The usual way of declaring an array is to simply line up the type name, followed by a variable name, followed by a size in brackets, as in this line of code: int Numbers[10]; This code declares an array of 10 integers. The first element gets index 0, and the final element gets index 9. =================================================================================== ========================================================= 98. Can abstract class have a constructor? Ans.--> 1)As we all know abstract classes also do have a constructor. So if we do not define any constructor inside the abstract class then JVM (Java Virtual Machine) will give a default constructor to the abstract class. 2)It must be declared with an abstract keyword. It can have a constructor, static method. =================================================================================== ============================================ 99. Define rules of Function overloading and function overriding? Ans.--> 1) Function Overloading happens in the same class when we declare same functions with different argument s in the same class. Function Overriding is happens in the child class when child class overrides parent class function. 2) In java, method overloading can't be performed by changing return type of the method only. Return type can be s ame or different in method overloading. But you must have to change the parameter. 3) Return type must be same or covariant in method overriding. =================================================================================== ============================================== 100. Explain the concept of Pure Virtual Functions? Ans.--> 1)A pure virtual function or pure virtual method is a virtual function that is required to be implemented by a derived class if the derived class is not abstract. Classes containing pure virtual methods are termed "abstract" and they cannot be instantiated directly. 2)Pure Virtual Function. A virtual function for which we are not required implementation is considered as pure virtu al function. For example, Abstract method in Java is a pure virtual function. =================================================================================== ==================================================