**1.Explain a virtual function?**

**Ans:**

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

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.

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.

**22. Explain a friend function?**

**Ans:**

A C++ friend functions are special functions which can access the private members of a class. They are considered to be a loophole in the Object Oriented Programming concepts, but logical use of them can make them useful in certain cases.

For instance: when it is not possible to implement some function, without making private members accessible in them. This situation arises mostly in case of operator overloading.

Or another example when you want to compare two private data members of two different classes in that case you need a common function which can make use of both the private variables of different class. In that case you create a normal function and make friend in both the classes, as to provide access of theirs private variables.

**23. Explain function overloading?**

**Ans:**

Function Overloading in Java **occurs when there are functions having the same name but have different numbers of parameters passed to it**, which can be different in data like int, double, float and used to return different values are computed inside the respective overloaded method.

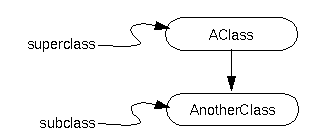
1. **class** Adder{
2. **static** **int** add(**int** a,**int** b){**return** a+b;}
3. **static** **int** add(**int** a,**int** b,**int** c){**return** a+b+c;}
4. }
5. **class** TestOverloading1{
6. **public** **static** **void** main(String[] args){
7. System.out.println(Adder.add(11,11));
8. System.out.println(Adder.add(11,11,11));
9. }

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

**Ans:**

**Subclasses, Superclasses, and Inheritance**

In Java, as in other object-oriented programming languages, classes can be derived from other classes. The derived class (the class that is derived from another class) is called a subclass. The class from which its derived is called the superclass.



In fact, in Java, all classes must be derived from some class. Which leads to the question "Where does it all begin?" The top-most class, the class from which all other classes are derived, is the Object class defined in java.lang. Object is the root of a hierarchy of classes.



The subclass inherits state and behavior in the form of variables and methods from its superclass. The subclass can just use the items inherited from its superclass as is, or the subclass can modify or override it. So, as you drop down in the hierarchy, the classes become more and more specialized:

**Definition:** A subclass is a class that derives from another class. A subclass inherits state and behavior from all of its ancestors. The term superclass refers to a class's direct ancestor as well as all of its ascendant classes.

Now would be a good time to review the discussion in [What Is Inheritance?(in the Writing Java Programs trail)](https://www.whitman.edu/mathematics/java_tutorial/java/objects/inheritance.html).

[**Creating Subclasses**](https://www.whitman.edu/mathematics/java_tutorial/java/javaOO/subclass.html)

To create a subclass of another class use the extends clause in your class declaration. ([The Class Declaration](https://www.whitman.edu/mathematics/java_tutorial/java/javaOO/classdecl.html) explains all of the components of a class declaration in detail.) As a subclass, your class inherits member variables and methods from its superclass. Your class can choose to hide variables or override methods inherited from its superclass.

[**Writing Final Classes and Methods**](https://www.whitman.edu/mathematics/java_tutorial/java/javaOO/final.html)

Sometimes, for security or design reasons, you want to prevent your class from being subclassed. Or, you may just wish to prevent certain methods within your class from being overriden. In Java, you can achieve either of these goals by marking the class or the method as *final*.

[**Writing Abstract Classes and Methods**](https://www.whitman.edu/mathematics/java_tutorial/java/javaOO/abstract.html)

On the other hand, some classes are written for the sole purpose of being subclassed (and are not intended to ever be instantiated). These classes are called *abstract classes* and often contain *abstract methods*.

[**The Object Class**](https://www.whitman.edu/mathematics/java_tutorial/java/javaOO/objectclass.html)

All objects in the Java environment inherit either directly or indirectly from the Object class. This section talks about the interesting methods in Object--methods that you may wish to invoke or override.

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

**Ans:**

**REFER Que 24**

**26. Explain an abstract class?**

**Ans**

An abstract class is a template definition of methods and variables of a class (category of objects) that contains one or more abstracted methods.

Points to Remember

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

**27. Explain operator overloading?**

**Ans:**

In C++, we can make operators work for user-defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Numbers, Fractional Numbers, Big Integer, etc.

Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.  
Example:

       int a;  
      float b,sum;  
      sum=a+b;

Here, variables “a” and “b” are of types “int” and “float”, which are built-in data types. Hence the addition operator ‘+’ can easily add the contents of “a” and “b”. This is because the addition operator “+” is predefined to add variables of built-in data type only.

Now, consider another example

class A  
{

};

int main()  
{  
      A   a1,a2,a3;

      a3= a1 + a2;

      return 0;  
}

In this example, we have 3 variables “a1”, “a2” and “a3” of type “class A”. Here we are trying to add two objects “a1” and “a2”, which are of user-defined type i.e. of type “class A” using the “+” operator. This is not allowed, because the addition operator “+” is predefined to operate only on built-in data types. But here, “class A” is a user-defined type, so the compiler generates an error. This is where the concept of “Operator overloading” comes in.   
Now, if the user wants to make the operator “+” to add two class objects, the user has to redefine the meaning of the “+” operator such that it adds two class objects. This is done by using the concept “Operator overloading”. So the main idea behind “Operator overloading” is to use c++ operators with class variables or class objects. Redefining the meaning of operators really does not change their original meaning; instead, they have been given additional meaning along with their existing ones.

#include<iostream>

**using** **namespace** std;

**class** Complex {

**private**:

**int** real, imag;

**public**:

    Complex(**int** r = 0, **int** i = 0) {real = r;   imag = i;}

    // This is automatically called when '+' is used with

    // between two Complex objects

    Complex operator + (Complex **const** &obj) {

         Complex res;

         res.real = real + obj.real;

         res.imag = imag + obj.imag;

**return** res;

    }

**void** print() { cout << real << " + i" << imag << '\n'; }

};

**int** main()

{

    Complex c1(10, 5), c2(2, 4);

    Complex c3 = c1 + c2;

    c3.print();

}

**Can we overload all operators?**   
Almost all operators can be overloaded except a few. Following is the list of operators that cannot be overloaded.

**Operators that can be overloaded**

1. Binary Arithmetic     ->     +, -, \*, /, %
2. Unary Arithmetic     ->     +, -, ++, —
3. Assignment     ->     =, +=,\*=, /=,-=, %=
4. Bit- wise      ->     & , | , << , >> , ~ , ^
5. De-referencing     ->     (->)
6. Dynamic memory allocation and De-allocation     ->     New, delete
7. Subscript     ->     [ ]
8. Function call     ->     ()
9. Logical      ->     &,  | |, !
10. Relational     ->     >, < , = =, <=, >=

there are some operators that cannot be overloaded. They are

* Scope resolution operator                                : :
* Member selection operator
* Member selection through                                   \*

Pointer to member variable

* Conditional operator                                         ? :
* Sizeof operator                                             sizeof()

**Why can’t the above-stated operators be overloaded?**

1. sizeof – This returns the size of the object or datatype entered as the operand. This is evaluated by the compiler and cannot be evaluated during runtime. The proper incrementing of a pointer in an array of objects relies on the sizeof operator implicitly. Altering its meaning using overloading would cause a fundamental part of the language to collapse.

2. typeid: This provides a CPP program with the ability to recover the actual derived type of the object referred to by a pointer or reference. For this operator, the whole point is to uniquely identify a type. If we want to make a user-defined type ‘look’ like another type, polymorphism can be used but the meaning of the typeid operator must remain unaltered, or else serious issues could arise.

3. Scope resolution (::): This helps identify and specify the context to which an identifier refers by specifying a namespace. It is completely evaluated at runtime and works on names rather than values. The operands of scope resolution are note expressions with data types and CPP has no syntax for capturing them if it were overloaded. So it is syntactically impossible to overload this operator.

4. Class member access operators (.(dot), .\* (pointer to member operator)): The importance and implicit use of class member access operators can be

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

**Ans:**

## What is Call by Value?

In this particular parameter passing method, the values of the actual parameters copy into the function’s formal parameters. It stores both types of parameters in different memory locations. Thus, if one makes any changes inside the function- it does not show on the caller’s actual parameters.

This method passes a copy of an actual argument to the formal argument of any called function. In the case of a Call by Value method, any changes or alteration made to the formal arguments in a called function does not affect the overall values of an actual argument. Thus, all the actual arguments stay safe, and no accidental modification occurs to them.

## What is Call by Reference?

In this case, both the formal and actual parameters refer to a similar location. It means that if one makes any changes inside the function, it gets reflected in the caller’s actual parameters.

This method passes the address or location of the actual arguments to the formal arguments of any called function. It means that by accessing the actual argument’s addresses, one can easily alter them from within the called function. Thus, in Call by Reference, it is possible to make alterations to the actual arguments. Thus, the code needs to handle the arguments very carefully. Or else, there might be unexpected results and accidental errors.

Difference Between Call by Value and Call by Reference

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Call By Value** | **Call By Reference** |
| Convention of Naming | In this case, the parameter’s **value** passes for invoking the function. In the case of calling a function, we pass the values of variables directly to a function. Thus, it has its name as Call by Value. | In this case, the parameter’s **reference** passes for the function invocation. Whenever calling a function, rather than passing the variables’ values, we pass its address instead (location of variables) to the function. Thus, it has its name as Call by Reference. |
| Effects of Changes | It copies the value of a passed parameter into the function’s argument. Thus, any changes that occur in any argument inside the function have no reflection on the passed parameter. | Both the passed parameter and the argument refer to a similar location. Thus, any changes occurring in any argument inside the function also reflects in the passed parameter. |
| Type of Passing | The method of Call by Value passes a **copy of the variable**. Here, the values of all the variables copy into their corresponding dummy variables, also called functions. | The method of Call by Reference passes the **variable itself**. Here, it copies the address of the actual variables in the calling function into the dummy variables called functions. |
| Memory Location (Referred) | The memory location referred to by the actual arguments and passed parameters of a function are different. Meaning, it creates the formal and actual arguments in different memory locations. | The memory location referred to by the actual arguments and passed parameters of a function are the same. Meaning, it creates the formal and actual arguments in the very same memory location. |
| Language Supported | Languages like C++, C#. PHP, Visual Basic NET, etc., provide support to the Call by Value and use it as their default method. | Only the JAVA language supports the Call by Reference method in programming. |
| Value Modification | In the Call by Value method, there is no modification in the original value. | In the Call by Reference method, there is a modification in the original value. |
| Internal Implementation | In the case of Call by Value, when we pass the value of the parameter during the calling of the function, it copies them to the function’s actual local argument. | In the case of Call by Reference, when we pass the parameter’s location reference/address, it copies and assigns them to the function’s local argument. Thus, both the actual argument and passed parameters refer to one similar location. |
| Method of Passing | The values of the variables in a Call by Value method pass using a straightforward method or a Simple technique. | Defining the pointer variables in a Call by Reference method is a prerequisite for storing the address of all the variables. |
| Manipulation of Variables | Using this method, any changes occurring to the dummy variables in any called function have no effect on the actual variable’s values in the calling function. Thus, you cannot alter or manipulate the actual variables’ values using the function calls. | Using this method, one can directly use the addresses to access the actual variables. Thus, any user can easily alter or manipulate the variables’ values through the function calls. |

**29. Explain the super keyword?**

**Ans:**

super is **used to refer immediate parent class instance variable**. We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

**30. 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