**Assignment**

1. **What is Inheritance in Java?**

**Ans:** When we construct a new class from existing class in such a way that the new class access all the features and properties of existing class called inheritance.

1. **What is superclass and subclass?**

**Ans:**

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

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

1. **How is Inheritance implemented/achieved in Java?**

**Ans:** Inheritance can be implemented or achieved by using **two keywords**:

**extends:** extends is a keyword that is used for developing the inheritance between two classes and two interfaces.

**implements:** implements keyword is used for developing the inheritance between a class and interface.

1. **What is polymorphism?**

**Ans:** Polymorphism is the greek word whose meaning is “same object having different behaviour”.

Customer

**Example:**

Student

friend

Teacher

void person(Teacher)

void person(Student)

void person(Friend)

void person(Customer)

Polymorphism in OOP is the ability of an entity to take several forms. In other words, it refers to the ability of an object (or a reference to an object) to take different forms of objects. It allows common data-gathering message to be sent to each class. Polymorphism encourages what is called ‘extendibility’ which means an object or a class can have its uses extended.

1. **Differentiate between method overloading and overriding?**

**Ans:**

|  |  |  |
| --- | --- | --- |
|  | **Method Overloading** | **Method Overriding** |
| 1) | Method overloading is used to increase the readability of the program. | Method overriding is used to provide the specific implementation of the method that is already provided by its super class. |
| 2) | Method overloading is performed within class. | Method overriding occurs in two classes that have IS-A (inheritance) relationship. |
| 3) | In case of method overloading, parameter must be different. | In case of method overriding, parameter must be same. |
| 4) | Method overloading is the example of compile time polymorphism. | Method overriding is the example of run time polymorphism. |
| 5) | In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter. | Return type must be same or covariant in method overriding. |

1. **What is an abstraction explained with an Example?**

**Ans:** Abstraction is a process of hiding the implementation details from the user, only the highlighted set of services provided to the user.

Example:

/\* abstract class program\*/

abstract class animal

{

animal()

{

System.out.println("All animal....!");

}

public abstract void sound();

}

class Dog extends animal

{

Dog()

{

super();

}

public void sound()

{

System.out.println("Dog is Barking.");

}

}

class Lion extends animal

{

Lion()

{

super();

}

public void sound()

{

System.out.println("Dog is Roar.");

}

}

class AbstractTest

{

public static void main(String[] args)

{

Dog d=new Dog();

Lion l=new Lion();

d.sound();

l.sound();

}

}

//Output:

// All animal....!

// All animal....!

// Dog is Barking.

// Dog is Roar.

1. **What is the difference between an abstract method and final method in Java? Explain with an example.**

**Ans:**

**Abstract method:-**a method which contain abstract modifier at the time of declaration is called abstract method.

It can only be used in abstract class. It doesn’t contain any body “{ }” and always ends with “ ;”

Abstract method must be overridden in sub classes otherwise it will also become a abstract class.

Whenever the action is common but implementation are different then we should use abstract

**Example:**

abstract class Programming

{

public abstract void Developer();

public abstract void Rank();

}

abstract class HTML extends Programming

{

@Override

public void Developer()

{

System.out.println("Tim Berners Lee");

}

}

class Java extends HTML

{

@Override

public void Rank()

{

System.out.println("Second Rank");

}

}

class AbstractMethodProgram

{

public static void main(String[] args)

{

Programming h=new Java();

h.Developer();

h.Rank();

}

}

//Output:

// Tim Berners Lee

// Second Rank

**Final method:-**Whenever we declare a method as a final it can’t be overridden to our extended class.

Sntax:-

final void m1()

{

……….

……….

}

Example:

**package** FinalMethod;

**class** A

{

**void** mNumber()

{

System.***out***.println("9685569893");

}

**final** **void** atmPIN()

{

System.***out***.println("1234");

}

}

**class** Thief **extends** A

{

@Override

**void** mNumber()

{

System.***out***.println("9685569893");

}

}

**public** **class** fullFinal

{

**public** **static** **void** main(String[] args)

{

Thief t=**new** Thief();

t.mNumber();

t.atmPIN();

}

}

//Output:

// 9685569893

/ 1234

1. **What is the final class in Java?**

**Ans:** Whenever we declare a class as a final it can’t be extended or inherited to sub classes.

Syntax:

final class A

{

………….

}

**Example**:

**final** **class** B

{

**void** mNumber()

{

System.***out***.println("9685569893");

}

**void** atmPIN()

{

System.***out***.println("1234");

}

}

**class** ghost **extends** B

{

@Override

**void** mNumber()

{

System.***out***.println("9685569893");

}

}

**public** **class** FinalClass1

{

**public** **static** **void** main(String[] args)

{

ghost t=**new** ghost();

t.mNumber();

t.atmPIN();

}

}

//Output:

// FinalClass1.java:15: error: cannot inherit from final B

// class ghost extends B

^

// 1 error

1. **Differentiate between abstraction and encapsulation?**

**Ans:**

|  |  |
| --- | --- |
| **Abstraction** | **Encapsulation** |
| 1. Abstraction is the process or method of gaining the information. | 1. While encapsulation is the process or method to contain the information. |
| 1. In abstraction, problems are solved at the design or interface level. | 1. While in encapsulation, problems are solved at the implementation level. |
| 1. Abstraction is the method of hiding the unwanted information. | 1. Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside. |
| 1. We can implement abstraction using abstract class and interfaces. | 1. Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public. |
| 1. In abstraction, implementation complexities are hidden using abstract classes and interfaces. | 1. While in encapsulation, the data is hidden using methods of getters and setters. |
| 1. The objects that help to perform abstraction are encapsulated. | 1. Whereas the objects that result in encapsulation need not be abstracted. |
| 1. Abstraction provides access to specific part of data. | 1. Encapsulation hides data and the user can not access same directly (data hiding. |
| 1. Abstraction focus is on “what” should be done. | 1. Encapsulation focus is on “How” it should be done. |

1. **Difference between Runtime and compile time polymorphism explain with an example?**

**Ans:**

|  |  |
| --- | --- |
| **Compile Time Polymorphism** | **Run time Polymorphism** |
| 1. In Compile time Polymorphism, the call is resolved by the compiler. | 1. In Run time Polymorphism, the call is not resolved by the compiler. |
| 1. It is also known as Static binding, Early binding and overloading as well. | 1. It is also known as Dynamic binding, Late binding and overriding as well. |
| 1. Method overloading is the compile-time polymorphism where more than one methods share the same name with different parameters or signature and different return type. | 1. Method overriding is the runtime polymorphism having the same method with same parameters or signature but associated with compared, different classes. |
| 1. It is achieved by function overloading and operator overloading. | 1. It is achieved by virtual functions and pointers. |
| 1. It provides fast execution because the method that needs to be executed is known early at the compile time. | 1. It provides slow execution as compare to early binding because the method that needs to be executed is known at the runtime. |
| 1. Compile time polymorphism is less flexible as all things execute at compile time. | 1. Run time polymorphism is more flexible as all things execute at run time. |
| 1. Inheritance is not involved. | 1. Inheritance is involved. |

**Example of Compile Time Polymorphism:-**

**package** polymorphism;

//compile time polymorphism will be achieved by method overloading(polymorphism means same object but different behavior)

**class** A

{

**void** add()

{

**int** a=10,b=20,c;

c=a+b;

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

}

**void** add(**int** x,**int** y)

{

**int** c;

c=x+y;

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

}

**void** add(**int** x,**double** y)

{

**double** c;

c=x+y;

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

}

}

**public** **class** CT {

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

// **TODO** Auto-generated method stub

A r=**new** A();

r.add();

r.add(100,200);

r.add(50,45.32);

}

}

//Output:

// 30

// 300

// 95.32

**Example of Run Time Polymorphism:-**

**package** polymorphism;

//Example of of Run Time Polymorphism.Using Overridding

**class** shape

{

**void** draw()

{

System.***out***.println("Can't Say shape Type");

}

}

**class** square **extends** shape

{

@Override

**void** draw()

{

**super**.draw();

System.***out***.println("Square shape");

}

}

**public** **class** RT1

{

**public** **static** **void** main(String[] args)

{

// **TODO** Auto-generated method stub

shape r=**new** square();

r.draw();

}

}

//Output:

// Can't Say shape Type

// Square shape