**Aim: Implement Scala programs to demonstrate class, objects and demonstrate inheritance**

**Theory:**

**1. Object**

Object is a real world entity. It contains state and behavior. Laptop, car, cell phone are the real world objects. Object typically has two characteristics:

**State**: data values of an object are known as its state.

**Behavior**: functionality that an object performs is known as its behavior.

Object in Scala is an instance of class. It is also known as runtime entity.

**2. Class**

Class is a template or a blueprint. It is also known as collection of objects of similar type. In scala, a class can contain: Data member, Member method, Constructor, Block, Nested class Super class information etc.You **must initialize all instance variables** in the class. There **is no default scope**. If you don't specify access scope, it is public. There must be an object in which main method is defined. It provides starting point for your program. Here, we have created an example of class.

Scala Sample Example of Class

class Student {

var id:Int = 0; // All fields must be initialized

var name:String = null;

}

object MainObject{

def main(args:Array[String]){

var s = new Student() // Creating an object

println(s.id+" "+s.name);

}

}

Scala Sample Example2 of Class with Constructor

In scala, you can create class like this also. Here, constructor is created in class definition. This is called primary constructor.

**class** Student(id:**Int**, name:String){     // Primary constructor

**def** show(){

println(id+" "+name)

}

}

**object** MainObject{

**def** main(args:Array[String]){

**var** s = **new** Student(100,"Martin")   // Passing values to constructor

s.show()                // Calling a function by using an object

}

}

**Scala Example of class that maintains the records of students**

**class** Student(id:**Int**, name:String){

**def** getRecord(){

println(id+" "+name);

}

}

**object** MainObject{

**def** main(args: Array[String]){

**var** student1 = **new** Student(101,"Raju");

**var** student2 = **new** Student(102,"Martin");

student1.getRecord();

student2.getRecord();

}

}

**1.1 Scala Anonymous object**

In scala, you can create anonymous object. An object which has no reference name is called anonymous object. It is good to create anonymous object when you don't want to reuse it further.

Scala Anonymous Object Example

**class** Arithmetic{

**def** add(a:**Int**, b:**Int**){

**var** add = a+b;

println("sum = "+add);

}

}

**object** MainObject{

**def** main(args:Array[String]){

**new** Arithmetic().add(10,10);  //Example of Anonymous Object

}

}

**1.2 Scala Singleton Object**

**Singleton object** is an object which is **declared by using object keyword instead by class**. No object is required to call methods declared inside singleton object. In Scala, there is no static concept. So Scala creates a singleton object to provide entry point for your program execution.

If **you don't create singleton object**, your code will **compile successfully** but **will not produce any output**. Methods declared inside Singleton Object are accessible globally. A singleton object can extend classes and traits.

**1.3 Scala Singleton Object Example**

object Singleton {

def main(args:Array[String]){

SingletonObject.hello () // No need to create object.

}

}

object SingletonObject {

def hello(){

println("Hello, This is Singleton Object")

}

}

**1.4 Scala Companion Object**

In Scala, when you have a class with same name as singleton object, it is called companion class and the singleton object is called companion object.

The companion class and its companion object both must be defined in the same source file.

**Scala Companion Object Example**

class ComapanionClass{

def hello(){

println("Hello, this is Companion Class.")

}

}

object CompanoinObject{

def main(args:Array[String]){

new ComapanionClass().hello()

println("And this is Companion Object.")

}

}

3. **Scala Constructor**

In scala, if you don't specify primary constructor, compiler creates a constructor which is known as primary constructor. All the statements of class body treated as part of constructor. It is also known as default constructor.

**class** Student{

println("Hello from default constructor");

}

Scala provides a concept of primary constructor with the definition of class. You don't need to define explicitly constructor if your code has only one constructor. It helps to optimize code. You can create primary constructor with zero or more parameters.

Scala Primary Constructor Example

class Student(id:Int, name:String){

def showDetails(){

println(id+" "+name);

}

}

object MainObject{

def main(args:Array[String]){

var s = new Student(101,"Rama");

s.showDetails()

}

}

**3.1 Scala Secondary (auxiliary) Constructor**

You can create any number of auxiliary constructors in a class. You must call primary constructor from inside the auxiliary constructor. this keyword is used to call constructor from other constructor. When calling other constructor make it first line in your constructor.

Scala Secondary Constructor Example

class Student(id:Int, name:String){

var age:Int = 0

def showDetails(){

println(id+" "+name+" "+age)

}

def this(id:Int, name:String,age:Int){

this(id,name) // Calling primary constructor, and it is first line

this.age = age

}

}

object MainObject{

def main(args:Array[String]){

var s = new Student(101,"Rama",20);

s.showDetails()

}

}

**Scala Example: Constructor Overloading**

In scala, you can overload constructor. Let's see an example.

class Student(id:Int){

def this(id:Int, name:String)={

this(id)

println(id+" "+name)

}

println(id)

}

object MainObject{

def main(args:Array[String]){

new Student(101)

new Student(100,"India")

}

}

**3.2 Scala Method Overloading**

Scala provides method overloading feature which allows us to define methods of same name but having different parameters or data types. It helps to optimize code.

**class** Arithmetic{

**def** add(a:**Int**, b:**Int**){

**var** sum = a+b

println(sum)

}

**def** add(a:**Int**, b:**Int**, c:**Int**){

**var** sum = a+b+c

println(sum) }  }

**object** MainObject{

**def** main(args:Array[String]){

**var** a  = **new** Arithmetic();

a.add(10,10);

a.add(10,10,10);  }

}

**4. Scala Method Overloading Example by using Different Data Type**

**class** Arithmetic{

**def** add(a:**Int**, b:**Int**){

**var** sum = a+b

println(sum)  }

**def** add(a:**Double**, b:**Double**){

**var** sum = a+b

println(sum) }

}

**object** MainObject{

**def** main(args:Array[String]){

**var** b = **new** Arithmetic()

b.add(10,10)

b.add(10.0,20.0)  }

}

**5. Scala this**

In scala, this is a keyword and used to refer current object. You can call instance variables, methods, constructors by using this keyword.

**class** ThisExample{

**var** id:**Int** = 0

**var** name: String = ""

**def** this(id:**Int**, name:String){

this()

this.id = id

this.name = name

}

**def** show(){

println(id+" "+name)

}

}

**object** MainObject{

**def** main(args:Array[String]){

**var** t = **new** ThisExample(101,"Martin")

t.show()

}

}

**5.1 Scala Constructor Calling by using this keyword**

**class** Student(name:String){

**def** this(name:String, age:**Int**){

this(name)

println (name+" "+age)  }

}

**object** MainObject{

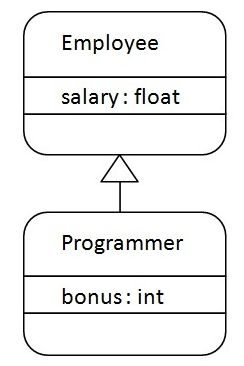
**def** main(args:Array[String]){

**var** s = **new** Student("Rama",100)  }

}

**6. Scala Inheritance**

Inheritance is an object oriented concept which is used to reusability of code. You can achieve inheritance by using extends keyword. To achieve inheritance a class must extend to other class. A class which is extended called super or parent class. a class which extends class is called derived or base class.



**Scala Single Inheritance Example**

**class** Employee{

**var** salary:Float = 10000

}

**class** Programmer **extends** Employee{

**var** bonus:**Int** = 5000

    println("Salary = "+salary)

    println("Bonus = "+bonus)

}

**object** MainObject{

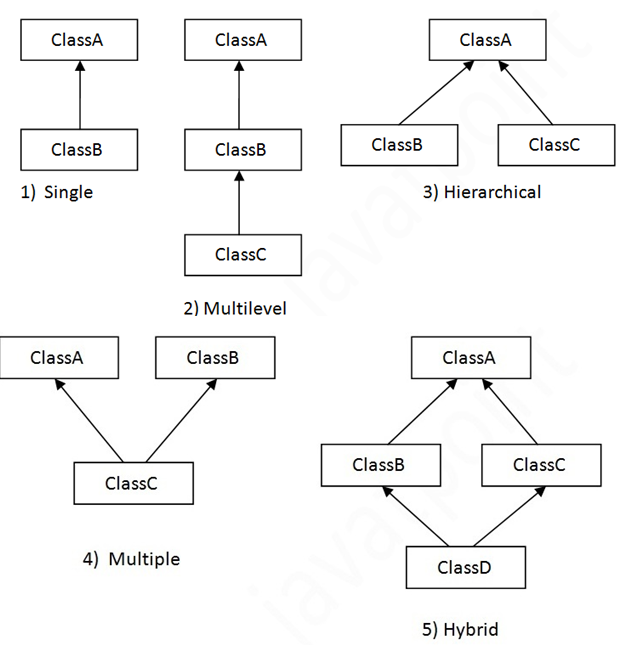
**def** main(args:Array[String]){

**new** Programmer()

    }

}

**6.2 Types of Inheritance in Scala**



**6.3 Scala Multilevel Inheritance Example**

**class** A{

**var** salary1 = 10000

}

**class** B **extends** A{

**var** salary2 = 20000

}

**class** C **extends** B{

**def** show(){

        println("salary1 = "+salary1)

        println("salary2 = "+salary2)

    }

}

**object** MainObject{

**def** main(args:Array[String]){{

**var** c = **new** C()

        c.show()

    }

}

**7. Scala Method Overriding**

When a subclass has the same name method as defined in the parent class, it is known as method overriding. When subclass wants to provide a specific implementation for the method defined in the parent class, it overrides method from parent class. In Scala, you must use either override keyword or override annotation to override methods from parent class.

Scala Method Overriding Example 1

**class** Vehicle{

**def** run(){

        println("vehicle is running")

    }

}

**class** Bike **extends** Vehicle{

     override **def** run(){

        println("Bike is running")

    }

}

**object** MainObject{

**def** main(args:Array[String]){

**var** b = **new** Bike()

        b.run()

    }

}

**7.1 Scala Method Overriding Example 2**

This example shows how subclasses override the method of parent class.

class Bank{

def getRateOfInterest()={

0

}

} ]

class SBI extends Bank{

override def getRateOfInterest()={

8

}

}

class ICICI extends Bank{

override def getRateOfInterest()={

7

}

}

class AXIS extends Bank{

override def getRateOfInterest()={

9

}

}

object MainObject{

def main(args:Array[String]){

var s=new SBI();

var i=new ICICI();

var a=new AXIS();

println("SBI Rate of Interest: "+s.getRateOfInterest());

println("ICICI Rate of Interest: "+i.getRateOfInterest());

println("AXIS Rate of Interest: "+a.getRateOfInterest());

}

}

**7.2 Scala Field Overriding Example1**

class Vehicle{

var speed:Int = 60

}

class Bike extends Vehicle{

var speed:Int = 100

def show(){

println(speed)

}

}

object MainObject{

def main(args:Array[String]){

var b = new Bike()

b.show()

}

}

8. **Scala Final**

Final is a keyword, which is used to prevent inheritance of super class members into derived class. You can declare final variables, methods and classes also.

**Scala Final Variable Example**

**You can't override final variables in subclass. Let's see an example.**

class Vehicle{

final val speed:Int = 60

}

class Bike extends Vehicle{

override val speed:Int = 100

def show(){

println(speed)

}

}

object MainObject{

def main(args:Array[String]){

var b = new Bike()

b.show()

}

}

9. Lab assignment to be completed in this session

1. Write a program to make a class called as Circle. It should have three methods namely: accept radius, calculate area and display the area.
2. Create a class employee with data member empid, empname, designation and salary. Write a methods get\_employee()-to take user input, show\_grade –to display grade of the employee based on salary.
3. Show employee () to display employee details.

|  |  |
| --- | --- |
| 1. Salary Range | 1. Grade |
| 1. <10000 | 1. D |
| 1. 10000-24999 | 1. C |
| 1. 25000-49999 | 1. B |
| 1. >50000 | 1. A |

1. Write a program to print the names of students by creating a Student class. If no name is passed while creating an object of Student class, then the name should be "Unknown", otherwise the name should be equal to the String value passed while creating object of Student class.
2. Five Bikers Compete in a race such that they Drive at Constant speed which may or may not be same as the other. To qualify the race, the speed of as racer must be more than the average speed of all 5 racers. Write Scala program to take as an input the speed of all racer and print back the speed of qualifying racer.
3. Create a class with a method that prints "This is parent class" and its subclass with another method that prints "This is child class". Now, create an object for each of the class and call

1 - method of parent class by object of parent class  
2 - method of child class by object of child class  
3 - method of parent class by object of child class

1. Create a class named 'Member' having the following members:  
   Data members  
   1 - Name  
   2 - Age  
   3 - Phone number  
   4 - Address  
   5 – Salary

It also has a method named 'printSalary' which prints the salary of the members.  
Two classes 'Employee' and 'Manager' inherits the 'Member' class. The 'Employee' and 'Manager' classes have data members 'specialization' and 'department' respectively. Now, assign name, age, phone number, address and salary to an employee and a manager by making an object of both of these classes and print the same.

1. Create a class named 'Rectangle' with two data members 'length' and 'breadth' and two methods to print the area and perimeter of the rectangle respectively. Its constructor having parameters for length and breadth is used to initialize length and breadth of the rectangle. Let class 'Square' inherit the 'Rectangle' class with its constructor having a parameter for its side (suppose s) calling the constructor of its parent class as 'super(s,s)'. Print the area and perimeter of a rectangle and a square.
2. Create a class named 'Shape' with a method to print "This is This is shape". Then create two other classes named 'Rectangle', 'Circle' inheriting the Shape class, both having a method to print "This is rectangular shape" and "This is circular shape" respectively. Create a subclass 'Square' of 'Rectangle' having a method to print "Square is a rectangle". Now call the method of 'Shape' and 'Rectangle' class by the object of 'Square' class.
3. Design a class hierarchy rooted in the class Employee that includes subclasses for HourlyEmployee and SalaryEmployee. The attributes shared in common by these classes include the name, and job title of the employee, plus the accessor and mutator methods needed by those attributes. The salaried employees need an attribute for weekly salary, and the corresponding methods for accessing and changing this variable. The hourly employees should have a pay rate and an hours worked variable. There should be an abstract method called calculateWeeklyPay (), defined abstractly in the superclass and implemented in the subclasses. The salaried worker's pay is just the weekly salary. Pay for an hourly employee is simply hours worked times pay rate.