Object Oriented Programming Final Term Exam (Spring-2020)

Submitted To: M. Ayub Khan

Submitted By: Aamir Saleem (12290)

Session: 8th Semester (Section-A) BS-SE

Date: 29th June 2020

Submitted By: Aamir Saleem (12290) Session: 8th Semester (Section-A) BS-SE

Q1. a. Why access modifiers are used in java, explain in detail Private and Default access modifiers?

Answer: Java modifiers are used to restrict access levels for variables, constructors, methods & classes. The access modifiers in Java specifies the accessibility of scope of a field, method or constructor. We can also change the access methods through Java modifiers.

Default Access Modifiers:

If there is no modifier used in Java, it is treated as default modifier as default. This modifier can only be accessed within the package. But it is more restrictive and protected than public modifier. Default Access Modifier means we do not explicitly declare any modifier. The default access modifier means that code inside the class itself as well as code inside classes in the same package as this class, can access the class, field, constructor or method which the default access modifier is assigned to. Therefore, the default access modifier is also sometimes referred to as the package access modifier.

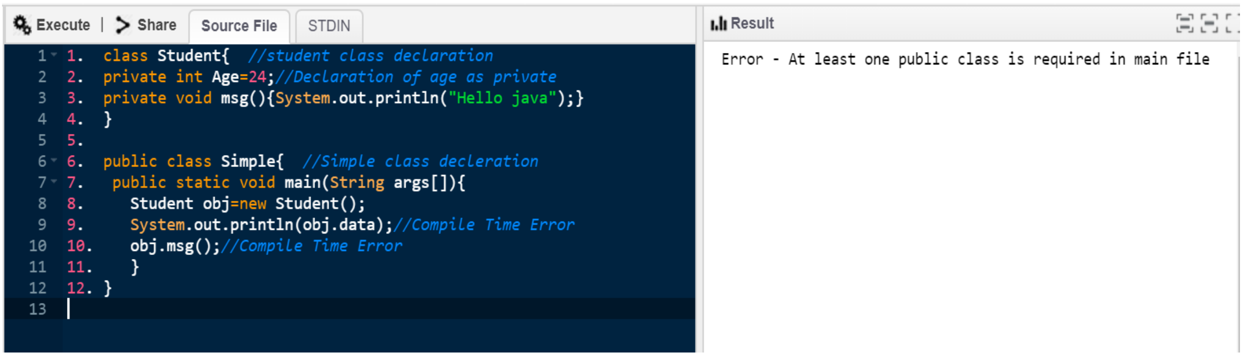
Private Access Modifier:

Methods, variables, and constructors that are declared private can only be accessed within the declared class itself. Private access modifier is the most restrictive access level. Class and interfaces cannot be private. Variables that are declared private can be accessed outside the class, if public getter methods are present in the class. Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world. If a method or variable is marked as private (has the private access modifier assigned to it), then only code inside the same class can access the variable, or call the method. Code inside subclasses cannot access the variable or method, nor can code from any external class. Classes cannot be marked with the private access modifier. Marking a class with the private access modifier would mean that no other class could access it, which means that you could not really use the class at all. Therefore, the private access modifier is not allowed for classes.

Q1. b. Write a specific program of the above-mentioned access modifiers in java.

Answer:

Example Code:



```
1.  class Student{  //student class declaration
2.  private int Age=24;//Declaration of age as private
3.  private void msg(){System.out.println("Hello java");}
4.  }
5.
6.  public class Simple{  //Simple class decleration
7.   public static void main(String args[]){
8.     Student obj=new Student();
9.     System.out.println(obj.data);//Compile Time Error
10.    obj.msg();//Compile Time Error
11.    }
12. }
```

```
Error - At least one public class is required in main file
```

Syntax:

1. class Student{ //student class declaration

2. private int Age=24;//Declaration of age as private

3. private void msg(){System.out.println("Hello java");}

4. }

5.

6. public class Simple{ //Simple class decleration

7. public static void main(String args[]){

8. Student obj=new Student();

9. System.out.println(obj.data);//Compile Time Error

10. obj.msg();//Compile Time Error

11. }

12. }

Note: Above program has been explained through comments.

---→

Q2. a. Explain in detail Public and Protected access modifiers?

Answer:

Public Access Modifiers:

Public Access Modifiers can be used for a class, method, constructor, interface, etc. Declared public modifier can be accessible from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe. However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses. A public class modifier has the widest scope among all other modifiers. The Java access modifier public means that all code can access the class, field, constructor or method, regardless of where the accessing code is located. The accessing code can be in a different class and different package.

Protected Access Modifiers:

The protected access modifier provides the same access as the default access modifier, with the addition that subclasses can access protected methods and member variables (fields) of the superclass. This is true even if the subclass is not located in the same package as the superclass. Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class. The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in an interface cannot be declared protected. Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it. The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class. It provides more accessibility than the default modifier

Q2. b. Write a specific program of the above-mentioned access modifiers in java.

<u>Answer:</u>

<u>Example of Public Access Modifier:</u>

<u>Syntax:</u>

```
//save by PAMA.java

package pack;
public class PAMA{
public void msg(){System.out.println("Hello Public access modifier ");}
}
//save by PAMB.java

package mypack;
import pack.*;

class PAMB{
  public static void main(String args[]){
   PAMA obj = new PAMA();
   obj.msg();
  }
}
```

<u>Output:</u> Hello Public access modifier

<u>Note:</u> Code is explained through comments.

<u>Example code of Protected Access Modifier:</u>

<u>Syntax:</u>

```
//save by ProtectedClassA.java
package pack;
public class ProtectedClassA {
protected void msg(){System.out.println("ProtectedClassA Hello");}
}
//save by ProtectedClassB.java
package mypack;
import pack.*;

class ProtectedClassB   extends ProtectedClassA{
  public static void main(String args[]){
   ProtectedClassB obj = new ProtectedClassB();
   obj.msg();
  }
}
```

<u>Output:</u> ProtectedClassA Hello

--→

Q3. a. What is inheritance and why it is used, discuss in detail?

Answer: Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

- The idea behind inheritance in Java is that we 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.

- Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Inheritance can be an effective method to share code between classes that have some traits in common, yet allowing the classes to have some parts that are different. The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

Terms Used in Inheritance:

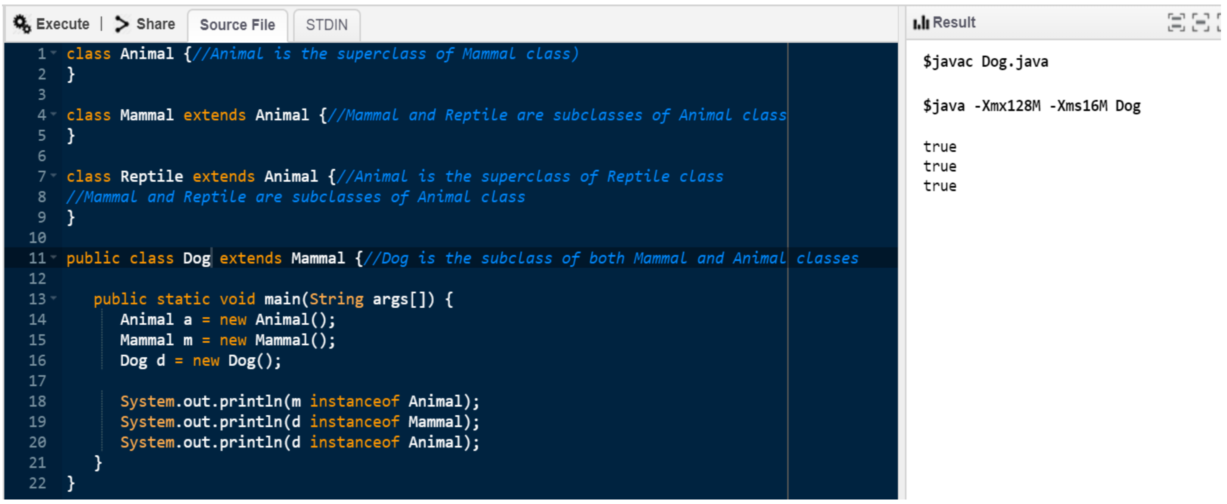
- *Class:* A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- *Sub Class/Child Class:* Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- *Super Class/Parent Class:* Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

- *Reusability:* As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Q3. b. b. Write a program using Inheritance class on Animal in java.

Answer:

Example Code:



```java
class Animal {//Animal is the superclass of Mammal class)
}

class Mammal extends Animal {//Mammal and Reptile are subclasses of Animal class
}

class Reptile extends Animal {//Animal is the superclass of Reptile class
//Mammal and Reptile are subclasses of Animal class
}

public class Dog extends Mammal {//Dog is the subclass of both Mammal and Animal classes

   public static void main(String args[]) {
      Animal a = new Animal();
      Mammal m = new Mammal();
      Dog d = new Dog();

      System.out.println(m instanceof Animal);
      System.out.println(d instanceof Mammal);
      System.out.println(d instanceof Animal);
   }
}
```

```
$javac Dog.java

$java -Xmx128M -Xms16M Dog

true
true
true
```

Syntax:

class Animal {//Animal is the superclass of Mammal class)

}

class Mammal extends Animal {//Mammal and Reptile are subclasses of Animal class

}

```
class Reptile extends Animal {//Animal is the superclass of Reptile class
//Mammal and Reptile are subclasses of Animal class
}

public class Dog extends Mammal {//Dog is the subclass of both Mammal and
Animal classes

  public static void main(String args[]) {
      Animal a = new Animal();
      Mammal m = new Mammal();
      Dog d = new Dog();

      System.out.println(m instanceof Animal);
      System.out.println(d instanceof Mammal);
      System.out.println(d instanceof Animal);
  }
}
```

<u>Output:</u>

```
$javac Dog.java
$java -Xmx128M -Xms16M Dog
true
true
true
```

---→

Q4. a. What is polymorphism and why it is used, discuss in detail?

Answer: The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Real-life Example: A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So, the same person possesses different behavior in different situations. This is called polymorphism.

Importance In OOP: Polymorphism is considered as one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word "poly" means many and "morphs" means forms, so it means many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object. It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed. The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object. A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Types of Polymorphism in Java:

- Compile time polymorphism:

  It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

- Runtime polymorphism:

  It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Q4. b. Write a program using polymorphism in a class on Employee in java?

Answer:

Example of Polymorphism on Employees:

Syntax:

```
/* File name : Employee.java */
public class Employee {
   private String name;
   private String address;
   private int number;
public Employee(String name, String address, int number) {
      System.out.println("Constructing an Employee");
      this.name = name;
      this.address = address;
      this.number = number;
   }
public void mailCheck() {
      System.out.println("Mailing a check to " + this.name + " " + this.address);
```

```
    }

    public String toString() {
        return name + " " + address + " " + number;
}

    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String newAddress) {
        address = newAddress;
    }
   public int getNumber() {
        return number;
    }
}

/* File name : Salary.java */
public class Salary extends Employee {
    private double salary; // Annual salary
    public Salary(String name, String address, int number, double salary) {
        super(name, address, number);
        setSalary(salary);
    }

    public void mailCheck() {
        System.out.println("Within mailCheck of Salary class ");
        System.out.println("Mailing check to " + getName()
        + " with salary " + salary);
    }
    public double getSalary() {
        return salary;
    }
     public void setSalary(double newSalary) {
        if(newSalary >= 0.0) {
            salary = newSalary;
        }
    }
     public double computePay() {
        System.out.println("Computing salary pay for " + getName());
        return salary/52;
    }
}
```

```
/* File name : VirtualDemo.java */
public class VirtualDemo {

   public static void main(String [] args) {
      Salary s = new Salary("Aamir Saleem", "Ambehta, UP", 3, 3600.00);
      Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);
      System.out.println("Call mailCheck using Salary reference --");
      s.mailCheck();
      System.out.println("\n Call mailCheck using Employee reference--");
      e.mailCheck();
   }
}
```

<u>Output:</u>

```
Constructing an Employee
Constructing an Employee

Call mailCheck using Salary reference --
Within mailCheck of Salary class
Mailing check to Aamir Saleem with salary 3600.0

Call mailCheck using Employee reference--
Within mailCheck of Salary class
Mailing check to John Adams with salary 2400.0
```

<u>--→</u>

Q5. a. Why abstraction is used in OOP, discuss in detail?

<u>Answer:</u>

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Example: A real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc. in the car. This is what abstraction is.

Importance: The reason why abstraction is considered as one of the important concepts is:

- It reduces the complexity of viewing things.
- Avoids code duplication and increases reusability.
- Helps to increase security of an application or program as only important details are provided to the user.

Methods of Abstract Class:

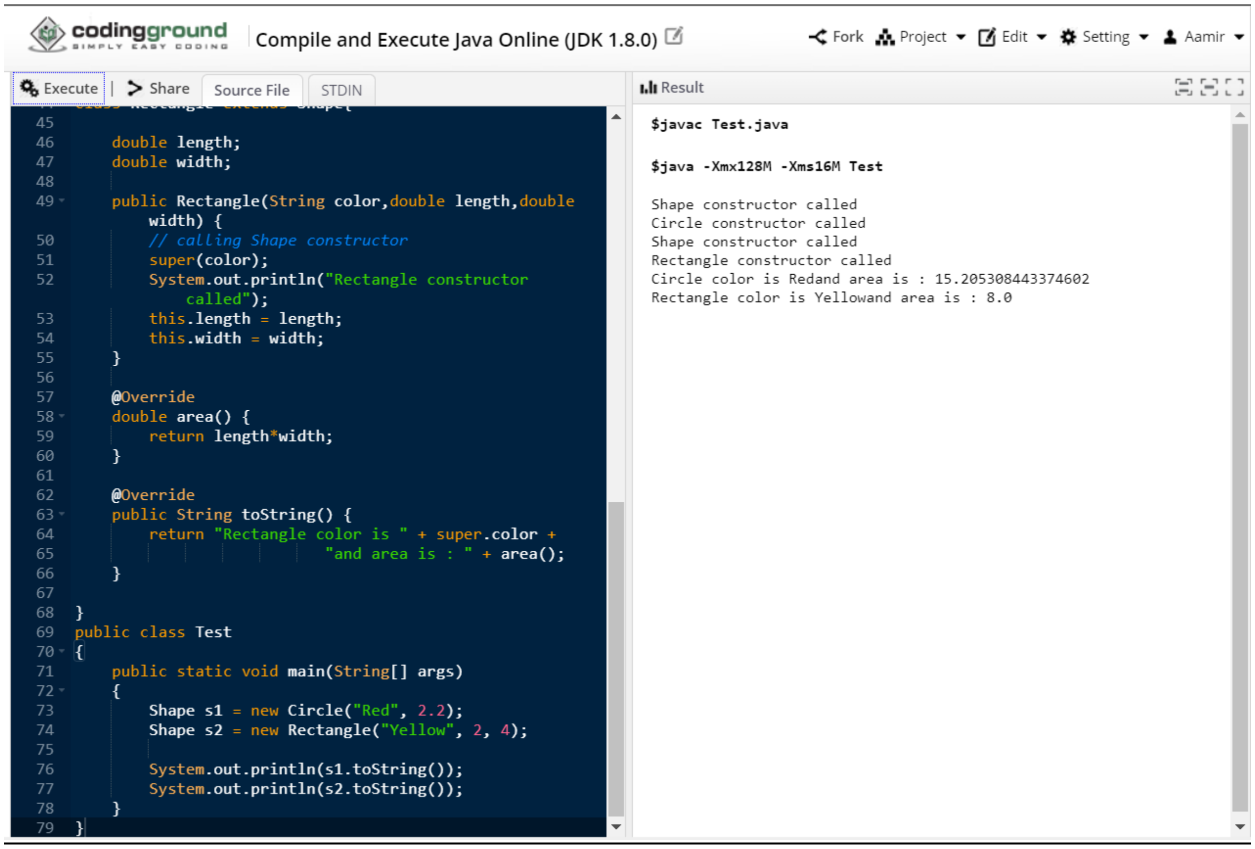
- An abstract class is a class that is declared with abstract keyword.
- An abstract method is a method that is declared without an implementation.
- An abstract class may or may not have all abstract methods. Some of them can be concrete methods
- A method defined abstract must always be redefined in the subclass, thus making overriding compulsory OR either make subclass itself abstract.
- Any class that contains one or more abstract methods must also be declared with abstract keyword.
- There can be no object of an abstract class. That is, an abstract class cannot be directly instantiated with the new operator.

- An abstract class can have parametrized constructors and default constructor is always present in an abstract class.

Q5. b. Write a program on abstraction in Java?

Answer:

Submitted By: Aamir Saleem (12290) Session: 8th Semester (Section-A) BS-SE



```java

    double length;
    double width;

    public Rectangle(String color,double length,double
        width) {
        // calling Shape constructor
        super(color);
        System.out.println("Rectangle constructor
            called");
        this.length = length;
        this.width = width;
    }

    @Override
    double area() {
        return length*width;
    }

    @Override
    public String toString() {
        return "Rectangle color is " + super.color +
                         "and area is : " + area();
    }

}
public class Test
{
    public static void main(String[] args)
    {
        Shape s1 = new Circle("Red", 2.2);
        Shape s2 = new Rectangle("Yellow", 2, 4);

        System.out.println(s1.toString());
        System.out.println(s2.toString());
    }
}
```

```
$javac Test.java

$java -Xmx128M -Xms16M Test

Shape constructor called
Circle constructor called
Shape constructor called
Rectangle constructor called
Circle color is Redand area is : 15.205308443374602
Rectangle color is Yellowand area is : 8.0
```

<u>Syntax:</u>

```
abstract class Shape
{
    String color;
      // these are abstract methods
    abstract double area();
    public abstract String toString();
       // abstract class can have constructor
    public Shape(String color) {
        System.out.println("Shape constructor called");
        this.color = color;
    }
     // this is a concrete method
    public String getColor() {
        return color;
    }
}
class Circle extends Shape
{
    double radius;

    public Circle(String color,double radius) {

        // calling Shape constructor
        super(color);
        System.out.println("Circle constructor called");
        this.radius = radius;
    }

    @Override
    double area() {
        return Math.PI * Math.pow(radius, 2);
    }

    @Override
    public String toString() {
        return "Circle color is " + super.color +
                        "and area is : " + area();
    }

}
class Rectangle extends Shape{

    double length;
    double width;

    public Rectangle(String color,double length,double width) {
        // calling Shape constructor
        super(color);
        System.out.println("Rectangle constructor called");
        this.length = length;
        this.width = width;
    }

    @Override
    double area() {
        return length*width;
    }

    @Override
    public String toString() {
        return "Rectangle color is " + super.color +
                              "and area is : " + area();
    }


}
public class Test
{
    public static void main(String[] args)
    {
        Shape s1 = new Circle("Red", 2.2);
        Shape s2 = new Rectangle("Yellow", 2, 4);

        System.out.println(s1.toString());
        System.out.println(s2.toString());
    }
}
```

Output:

```
$javac Test.java
$java -Xmx128M -Xms16M Test
Shape constructor called
Circle constructor called
Shape constructor called
Rectangle constructor called
Circle color is Red and area is : 15.205308443374602
Rectangle color is Yellow and area is : 8.0
```

Note: The above code is explained in comments.