Module – 6

- ## Object-Oriented Programming Concepts
  - ### Encapsulation
    - In Java, encapsulation is one of the core concepts of Object Oriented Programming (OOP) in which we bind the data members and methods into a single unit. Encapsulation is used to hide the implementation part and show the functionality for better readability and usability. The following are important points about encapsulation.
    - **Benefits**
      - Better Code Management
      - Simpler Parameter Passing
      - getter and setter
    - **uses**
      - Data Hiding
      - Data Integrity
      - Reusability
      - Security

  - ### Inheritance
    - Inheritance is a process where one class acquires the properties (methods and attributes) of another. With the use of inheritance, the information is made manageable in a hierarchical order.
    - 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).
    - Needs
      - **Code Reusability**
      - **Extensibility**
      - **Implantation of Method Overriding**
      - **Achieving Abstraction**
    - **Disadvantages**
      - Complexity
      - Tight Coupling
  - ### Polymorphism
    - one entity can take many forms
    - **key features**
      - Multiple Behaviors
      - Method Overriding
      - Method Overloading
      - Runtime Decision

- **Use**
  - Code Reusability
  - Flexibility
  - Abstraction
  - Dynamic Behavior
- **Types**
  - Compile-Time Polymorphism (Static)
  - Runtime Polymorphism (Dynamic)

➢ Abstraction

- Abstraction in Java is the process of hiding internal implementation details and showing only essential functionality to the user. It focuses on what an object does rather than how it does it.
- **Advantages**
  - It makes complex systems easier to understand by hiding the implementation details.
  - it keeps different part of the system separated.
  - It maintains code more efficiently.
  - It increases the security by only showing the necessary details to the user.
- **Disadvantages**
  - It can add unnecessary complexity if overused.
  - May reduce flexibility in implementation.
  - Makes debugging and understanding the system harder for unfamiliar users.
  - Overhead from abstraction layers can affect performance.

➢ Single Inheritance

- In single inheritance, a sub-class is derived from only one super class. It inherits the properties and behavior of a single-parent class. Sometimes, it is also known as simple inheritance.

```java
//Super class
class Vehicle {
    Vehicle() {
        System.out.println("This is a Vehicle");
    }
}

// Subclass
class Car extends Vehicle {
    Car() {
        System.out.println("This Vehicle is Car");
    }
}

public class Test {
    public static void main(String[] args) {
        // Creating object of subclass invokes base class
constructor
        Car obj = new Car();
    }
}
```

➢ Multilevel inheritance

- In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also acts as the base class for other classes.

```
class Vehicle {
    Vehicle() {
        System.out.println("This is a Vehicle");
    }
}
class FourWheeler extends Vehicle {
    FourWheeler() {
        System.out.println("4 Wheeler Vehicles");
    }
}
class Car extends FourWheeler {
    Car() {
        System.out.println("This 4 Wheeler Vehicle is a Car");
    }
}
public class Geeks {
    public static void main(String[] args) {
        Car obj = new Car(); // Triggers all constructors in
order
    }
}
```

➢ Hierarchical Inheritance

- In hierarchical inheritance, more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class. For example, cars and buses both are vehicle.

```
class Vehicle {
    Vehicle() {
        System.out.println("This is a Vehicle");
    }
}

class Car extends Vehicle {
    Car() {
        System.out.println("This Vehicle is Car");
    }
}

class Bus extends Vehicle {
    Bus() {
        System.out.println("This Vehicle is Bus");
    }
}

public class Test {
    public static void main(String[] args) {
        Car obj1 = new Car();
        Bus obj2 = new Bus();
    }
}
```

➢ Method Overriding

- Overriding in Java occurs when a subclass or child class implements a method that is already defined in the superclass or base class.

```java
// Example of Overriding in Java
class Animal {
    // Base class
    void move() { System.out.println(
      "Animal is moving."); }
    void eat() { System.out.println(
      "Animal is eating."); }
}

class Dog extends Animal {
    @Override void move()
    { // move method from Base class is overriden in this
      // method
        System.out.println("Dog is running.");
    }
    void bark() { System.out.println("Dog is barking."); }
}

public class Geeks {
    public static void main(String[] args)
    {
        Dog d = new Dog();
        d.move(); // Output: Dog is running.
        d.eat(); // Output: Animal is eating.
        d.bark(); // Output: Dog is barking.
    }
}
```

➢ Dynamic Method Dispatch

- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

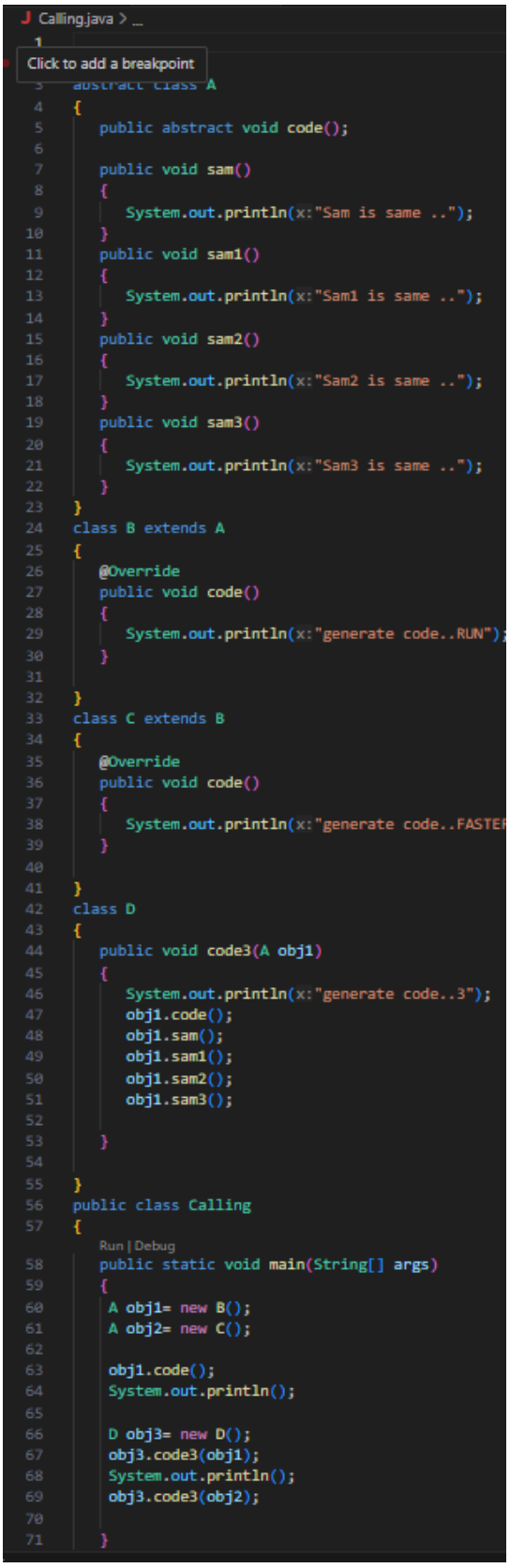
Lab Exercise:

1. **Single Inheritance**

encapsulation.java

```
2
3  public class encapsulation {
4      public static void main(String[] args)
5      {
6          B obj =  new B();
7          obj.create();
8
9          obj.create(3);
10
11         double x= (double)obj.multi(45, 54);
12         System.out.println(x);
13
14         double y=(double)obj.multi(6, 57,78);
15         System.out.println(y);
16
17         C obj1 = new C();
18         D obj2 = new D();
19         obj2.cr1(obj1);
20
21     }
22 }
23 class A
24 {
25     public void create()
26     {
27         System.out.println("creating ..A");
28     }
29     public int multi(int n1,int n2)
30     {
31         return n1*n2;
32     }
33     public int multi(int n1,double n2)
34     {
35         System.out.println("in double");
36         return (int) (n1*n2);
37     }
38 }
39 class B extends A
40 {
41     public void create(int a)
42     {
43         System.out.println("creating ..B");
44     }
45     public int multi(int n1,int n2,int n3)
46     {
47         return n1*n2*n3;
48     }
```

2. Multilevel Inheritance



3. Override by data type

encapsulation.java | fetching.java

```java
package core_JAva;

public class encapsulation {
    public static void main(String[] args)
    {
        B obj =  new B();
        obj.create();

        obj.create(3);

        double x= (double)obj.multi(45, 54);
        System.out.println(x);

        double y=(double)obj.multi(6, 57,78);
        System.out.println(y);

        C obj1 = new C();
        D obj2 = new D();
        obj2.cr1(obj1);

    }
}
class A
{
    public void create()
    {
        System.out.println("creating ..A");
    }
    public int multi(int n1,int n2)
    {
        return n1*n2;
    }
    public int multi(int n1,double n2)
    {
        System.out.println("in double");
        return (int) (n1*n2);
    }
}
class B extends A
{
    public void create(int a)
    {
        System.out.println("creating ..B");
    }
    public int multi(int n1,int n2,int n3)
    {
        return n1*n2*n3;
    }
}
class C
{
    public void cr()
    {
        System.out.println("creating ..C");

    }
}
```