Name: Sownthari R P

Date: 09.08.2024

1. Implement Abstract class with overloading and overriding

```
abstract class Animal {
    abstract void sound();

    void eat() {
        System.out.println("Animal is eating.");
    }

    void eat(String food) {
        System.out.println("Animal is eating " + food);
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks.");
    }

    @Override
    void eat() {
        System.out.println("Dog is eating.");
    }
}

class AnimalSound {
    private Animal animal;
    public AnimalSound(Animal animal) {
        this.animal = animal;
    }

    public void makeSound() {
        animal.sound();
    }

    public void makeEat() {
        animal.eat();
    }

    public void makeEat(String food) {
        animal.eat(food);
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        AnimalSound animalSound = new AnimalSound(myDog);

        animalSound.makeSound();
        animalSound.makeEat();
        animalSound.makeEat("bone");
    }
}
```

Output:

```
Dog barks.
Dog is eating.
```

```
Animal is eating bone.
```

2. Implement Multiple inheritance with Interface

```java
interface CanFly {
    void fly();
}

interface CanSwim {
    void swim();
}

class Duck implements CanFly, CanSwim {
    @Override
    public void fly() {
        System.out.println("Duck is flying.");
    }

    @Override
    public void swim() {
        System.out.println("Duck is swimming.");
    }
}

class AnimalActions {
    private CanFly flyer;
    private CanSwim swimmer;

    public AnimalActions(CanFly flyer, CanSwim swimmer) {
        this.flyer = flyer;
        this.swimmer = swimmer;
    }
    public void performFly() {
        flyer.fly();
    }

    public void performSwim() {
        swimmer.swim();
    }
}

public class Main {
    public static void main(String[] args) {
        Duck duck = new Duck();
        AnimalActions actions = new AnimalActions(duck, duck);

        actions.performFly();
        actions.performSwim();
    }
}
```

Output:

```
Duck is flying.
Duck is swimming.
```

3. Show final methods in the class that can't be overridden

```
class Animal {
    final void sleep() {
        System.out.println("Animal is sleeping.");
    }

    void eat() {
        System.out.println("Animal is eating.");
    }
}

class Dog extends Animal {
    @Override
    void eat() {
        System.out.println("Dog is eating.");
    }
}

class AnimalBehavior {
    private Animal animal;

    public AnimalBehavior(Animal animal) {
        this.animal = animal;
    }

    public void performSleep() {
        animal.sleep();
    }

    public void performEat() {
        animal.eat();
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        AnimalBehavior behavior = new AnimalBehavior(myDog);
        behavior.performSleep();
        behavior.performEat();
    }
}
```

Output:

```
Animal is sleeping.
Dog is eating.
```