Experiment No. 7
Implement a program using super and final keyword.
Date of Performance:11/09/24
Date of Submission:

Aim: To implement the concept of super and final keyword.

Objective: To understand the usage of superclass and final method , variables and class

Theory:

super and **final** keywords are two popular and useful keywords in Java. They also play a significant role in dealing with Java programs and their classes. In this chapter, you will learn about how to use super and final within a Java program.

Syntax:

```
super.<method-name>();
```

- Super variables refer to the variable of a variable of the parent class.
- Super() invokes the constructor of immediate parent class.
- Super refers to the method of the parent class

Instance refers an instance variable of the current class by default, but when you have to refer parent class instance variable, you have to use super keyword to distinguish between parent class (here employee) instance variable and current class (here, clerk) instance variable.

What is final in Java?

Final is a keyword in Java that is used to restrict the user and can be used in many respects. Final can be used with:

- Class
- Methods
- Variables

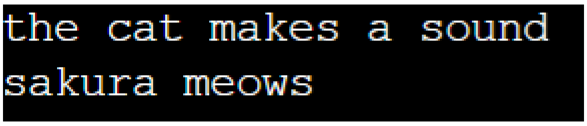
A method declared as final cannot be overridden; this means even when a child class can call the final method of parent class without any issues, but the overriding will not be possible.

Once a variable is assigned with the keyword final, it always contains the same exact value. Again things may happen like this; if a final variable holds a reference to an object then the state of the object can be altered if programmers perform certain operations on those objects, but the variable will always refer to the same object. A final variable that is not initialized at the time of declaration is known as a blank final variable. If you are declaring a final variable in a constructor, then you must initialize the blank final variable within the constructor of the class. Otherwise, the program might show a compilation error.

Code:

```
//super keyword
class Animal {
   String name;

   Animal(String name) {
      this.name = name;
   }

   void makeSound() {
      System.out.println("the cat makes a sound");
   }
}

class Cat extends Animal {

   Cat(String name) {
      super(name);
   }


   void makeSound() {
      super.makeSound();
      System.out.println(name + " meows");
   }
}

public class Super{
   public static void main(String[] args) {
      Cat cat = new Cat("sakura");
      cat.makeSound();
   }
}
```

Output:

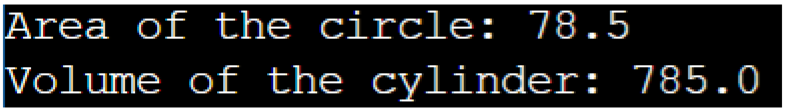


```
the cat makes a sound
sakura meows
```

//final keyword

```
final class Constants {
   static final double PI = 3.14;
}

class Circle {
   private double radius;

   Circle(double radius) {
      this.radius = radius;
   }

   final double area() {
      return Constants.PI * radius * radius;
   }
}

class Cylinder extends Circle {
   private double height;

   Cylinder(double radius, double height) {
      super(radius);
      this.height = height;
   }

   double volume() {
      return area() * height;
   }
}

public class Final{
   public static void main(String[] args) {
      Circle circle = new Circle(5);
      System.out.println("Area of the circle: " + circle.area());

      Cylinder cylinder = new Cylinder(5, 10);
      System.out.println("Volume of the cylinder: " + cylinder.volume());
   }
}
```

Output:



```
Area of the circle: 78.5
Volume of the cylinder: 785.0
```

Conclusion:

In conclusion, the two Java programs effectively illustrate the use of the `super` and `final` keywords, showcasing important object-oriented programming concepts. The first program demonstrates how `super` allows derived classes to access superclass constructors and methods, facilitating code reuse and method overriding. The second program highlights the `final` keyword's role in preventing modifications, ensuring constants remain unchanged, and restricting method overriding and class inheritance. Together, these examples reinforce fundamental principles of encapsulation, inheritance, and immutability in Java programming.