Abstraction:

- It is the process of hiding the implementation details from the user and providing only the relevant information to the user.
- It allows you to represent an object's essential features without revealing its complexities.
- It allows us to focus on what the object does instead of how it does it.
- Simply, you create a blueprint of an object without specifying how it works internally.
- Abstraction in Java is achieved in 2 ways:
  - Abstract class
  - Interface

Abstract  Class

- An abstract class is a class that cannot be instantiated on its own.
- An abstract class is a class that contains abstract keyword in its declaration.
- It can contain both abstract and non-abstract (concrete) methods.
- Abstract classes are designed to be subclassed and may have abstract methods that must be implemented in their subclasses.
- Abstract classes provide a way to define a common interface for a group of related classes while leaving the details of their implementation to the subclasses.
- An abstract class cannot be instantiated. i.e. object cannot be created for an abstract class.

  General form:
  ```
   abstract class classname{
  //block of codes
  }
  ```

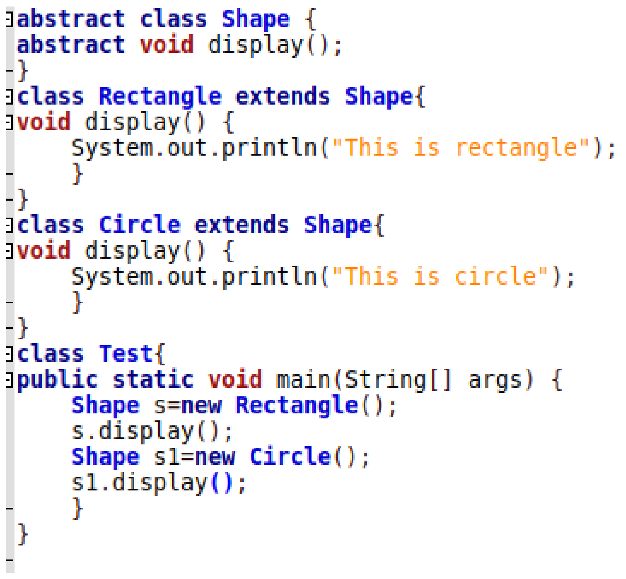
Abstract Methods

- If you want a class to contain a particular method but want implementation of that method to be determined by child classes, you can declare method in parent class as abstract.
- An abstract method is a method declared in an abstract class without providing any implementation.
- In abstract method, abstract keyword is used before method name in method declaration.
- It does not contain the method body but only contains the method signature (method name, parameters, and return type).
- Also, abstract method ends with semicolon.
- Subclasses of an abstract class must provide concrete implementations for all the abstract methods defined in the parent abstract class.

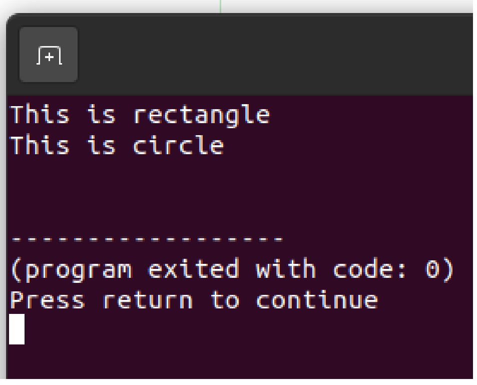
Simple form:

```
Public abstract class Teacher {
        String name;
        String subject;
        public abstract void teach();
        }
```

Program 1: Illustrate concept of abstract class and abstract methods.

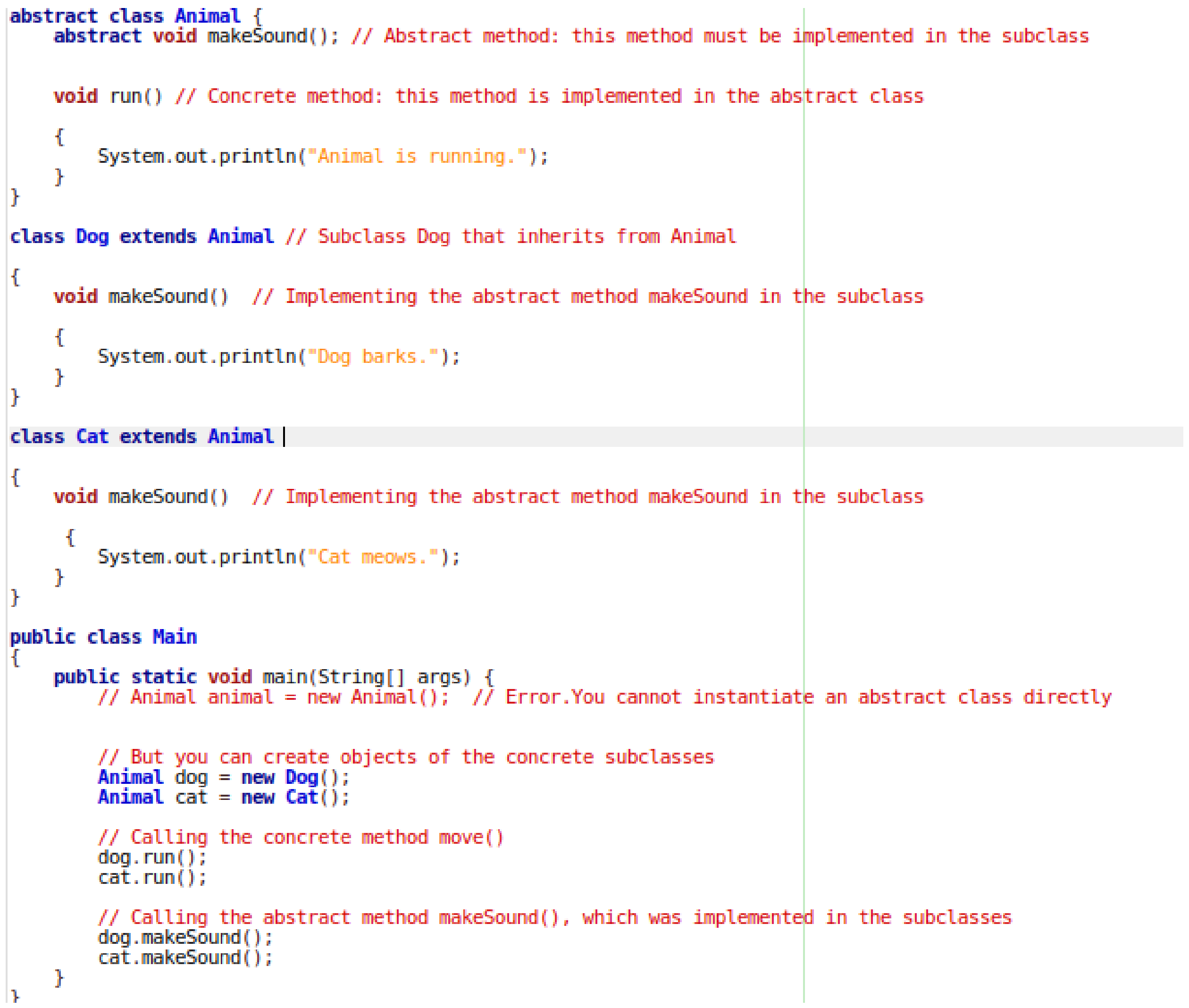


```
abstract class Shape {
abstract void display();
}
class Rectangle extends Shape{
void display() {
    System.out.println("This is rectangle");
    }
}
class Circle extends Shape{
void display() {
    System.out.println("This is circle");
    }
}
class Test{
public static void main(String[] args) {
    Shape s=new Rectangle();
    s.display();
    Shape s1=new Circle();
    s1.display();
    }
}
```



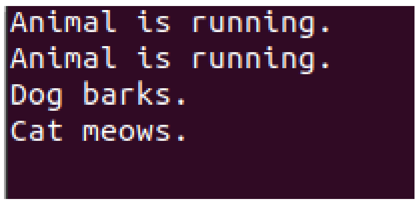
```
This is rectangle
This is circle


-------------------
(program exited with code: 0)
Press return to continue
```

Program 2:



```java
abstract class Animal {
    abstract void makeSound(); // Abstract method: this method must be implemented in the subclass


    void run() // Concrete method: this method is implemented in the abstract class

    {
        System.out.println("Animal is running.");
    }
}

class Dog extends Animal // Subclass Dog that inherits from Animal

{
    void makeSound()  // Implementing the abstract method makeSound in the subclass

    {
        System.out.println("Dog barks.");
    }
}

class Cat extends Animal

{
    void makeSound()  // Implementing the abstract method makeSound in the subclass

     {
        System.out.println("Cat meows.");
    }
}

public class Main
{
    public static void main(String[] args) {
        // Animal animal = new Animal();  // Error.You cannot instantiate an abstract class directly


        // But you can create objects of the concrete subclasses
        Animal dog = new Dog();
        Animal cat = new Cat();

        // Calling the concrete method move()
        dog.run();
        cat.run();

        // Calling the abstract method makeSound(), which was implemented in the subclasses
        dog.makeSound();
        cat.makeSound();
    }
}
```

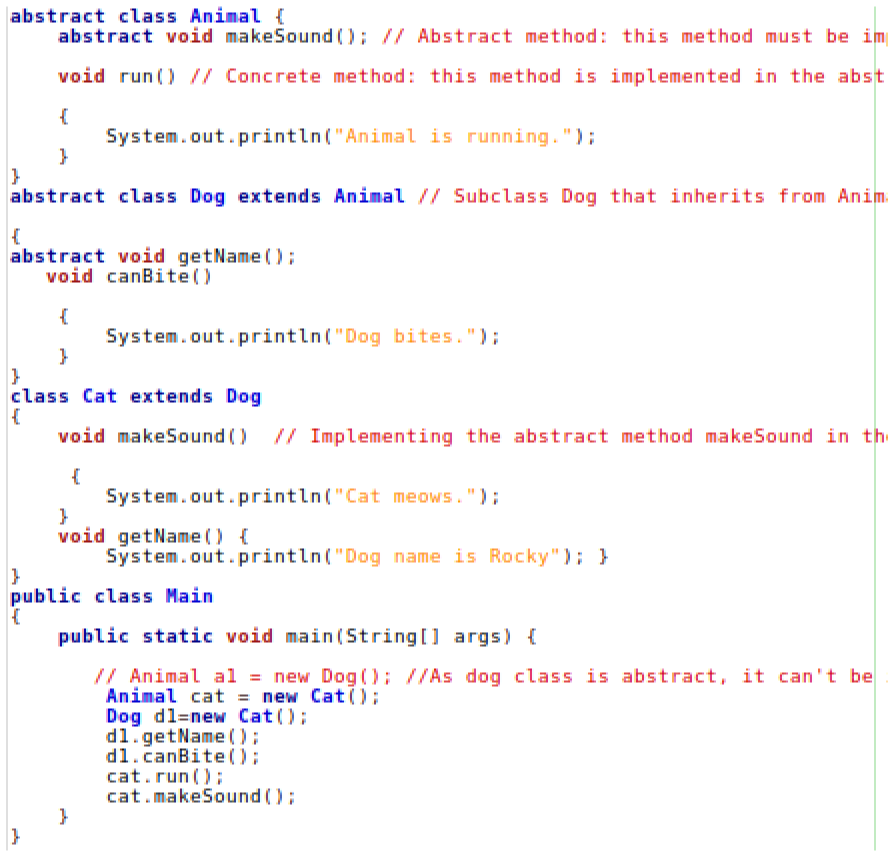
Output:



```
Animal is running.
Animal is running.
Dog barks.
Cat meows.
```

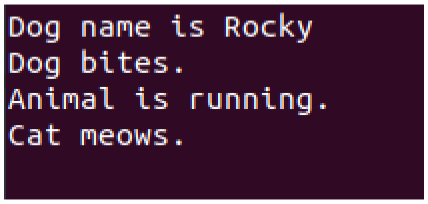
Note:

If the dog class is not able to implement methods of Animal class(abstract class), Dog class must be made abstract as:



```java
abstract class Animal {
    abstract void makeSound(); // Abstract method: this method must be implemented in the subclass

    void run() // Concrete method: this method is implemented in the abstract class

    {
        System.out.println("Animal is running.");
    }
}
abstract class Dog extends Animal // Subclass Dog that inherits from Animal

{
abstract void getName();
   void canBite()

    {
        System.out.println("Dog bites.");
    }
}
class Cat extends Dog
{
    void makeSound()  // Implementing the abstract method makeSound in the subclass

     {
        System.out.println("Cat meows.");
    }
    void getName() {
        System.out.println("Dog name is Rocky"); }
}
public class Main
{
    public static void main(String[] args) {

       // Animal a1 = new Dog(); //As dog class is abstract, it can't be instantiated
        Animal cat = new Cat();
        Dog d1=new Cat();
        d1.getName();
        d1.canBite();
        cat.run();
        cat.makeSound();
    }
}
```

Output:



```
Dog name is Rocky
Dog bites.
Animal is running.
Cat meows.
```

**Interface:**

- Interface is the group of methods and fields declarations with a name.
- Interface donot provide any code that implement those methods.
- Interface cannot be instantiated directly like abstract class.
- The class which implements the interface provides implementation for each of methods that the interface defines.
- It is defined with ‘interface’ keyword.

- Access specifier is either public or default.

Syntax:

```
        Interface interfacename
        {
                variable declaration;
                method declarations;
        }
```

- Variable declaration in interface is always constant though it may or maynot be declared as static/final.
- Eg. static final type variablename=value;
  Or, type variablename=value;

Method declaration:

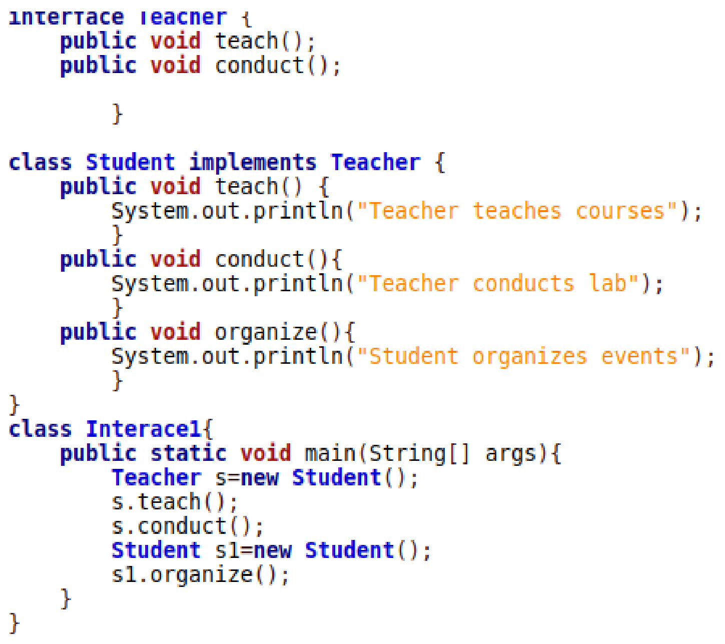
```
  returntype methodname(parameter_list);
```

Note:
Interface methods are by default abstract and public.

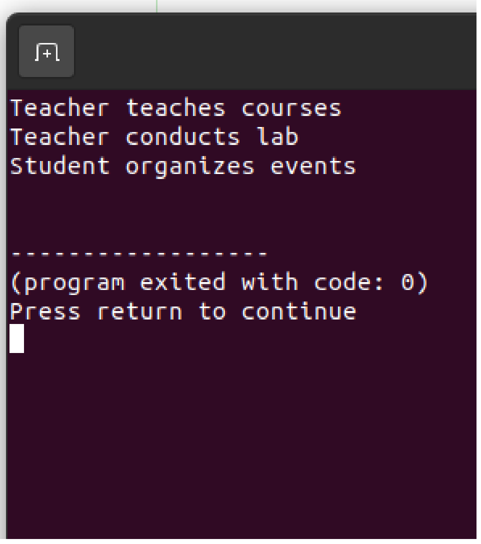
Simple form:

```
 interface Student {
        final static float id=1;
        void getData(String name, int roll);
        void display();
        }
```

Program Example1:



```
interface Teacher {
    public void teach();
    public void conduct();

        }

class Student implements Teacher {
    public void teach() {
        System.out.println("Teacher teaches courses");
        }
    public void conduct(){
        System.out.println("Teacher conducts lab");
        }
    public void organize(){
        System.out.println("Student organizes events");
        }
}
class Interace1{
    public static void main(String[] args){
        Teacher s=new Student();
        s.teach();
        s.conduct();
        Student s1=new Student();
        s1.organize();
    }
}
```



```
Teacher teaches courses
Teacher conducts lab
Student organizes events


------------------
(program exited with code: 0)
Press return to continue
```

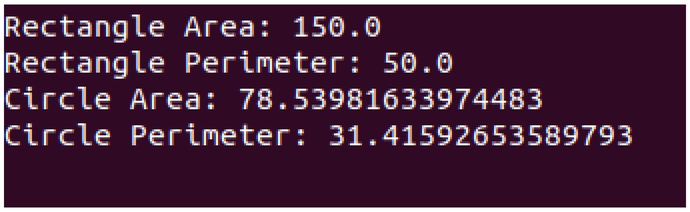
Example 2: Create an interface called Shape that specifies two methods: calculateArea() and calculatePerimeter(). Different shapes, like Rectangle and Circle, will implement this Shape interface to provide their own implementations of these methods.

```
interface Shape {
   double calculateArea();
   double calculatePerimeter();
}
class Rectangle implements Shape {
   double length;
   double width;
   public Rectangle(double length, double width) {
      this.length = length;
      this.width = width;
   }
   public double calculateArea() {
      return length * width;
```

```
    }
    public double calculatePerimeter() {
        return 2 * (length + width);
    }
}
class Circle implements Shape {
     double radius;

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

    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    public double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }
}
public class MainInterfaceExample {
    public static void main(String[] args) {
        Shape rects1 = new Rectangle(15, 10);
        Shape circles2 = new Circle(5);

        System.out.println("Rectangle Area: " + rects1.calculateArea());
        System.out.println("Rectangle Perimeter: " +
rects1.calculatePerimeter());

        System.out.println("Circle Area: " + circles2.calculateArea());
        System.out.println("Circle Perimeter: " +
circles2.calculatePerimeter());
```

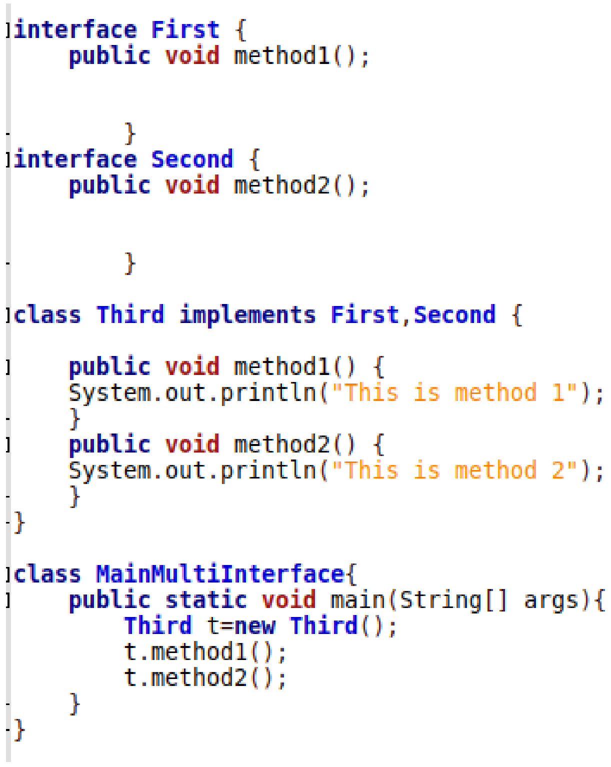
```
    }
}
```

Output:

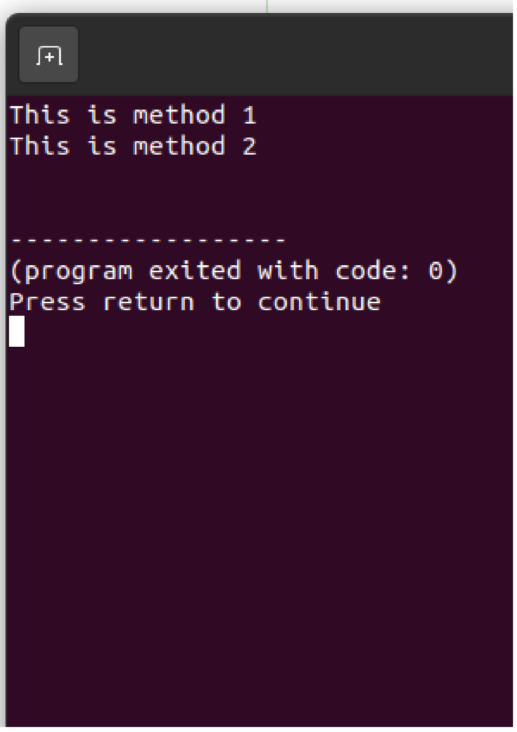


```
Rectangle Area: 150.0
Rectangle Perimeter: 50.0
Circle Area: 78.53981633974483
Circle Perimeter: 31.41592653589793
```

Program Example showing Multiple interface:



```
interface First {
    public void method1();


        }
interface Second {
    public void method2();


        }

class Third implements First,Second {

    public void method1() {
    System.out.println("This is method 1");
    }
    public void method2() {
    System.out.println("This is method 2");
    }
}

class MainMultiInterface{
    public static void main(String[] args){
        Third t=new Third();
        t.method1();
        t.method2();
    }
}
```



```
This is method 1
This is method 2


------------------
(program exited with code: 0)
Press return to continue
```