Java Abstraction

Abstract Classes and Methods

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.
Abstraction can be achieved with either **abstract classes** or **interfaces**

The abstract keyword is a non-access modifier, used for classes and methods:

- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

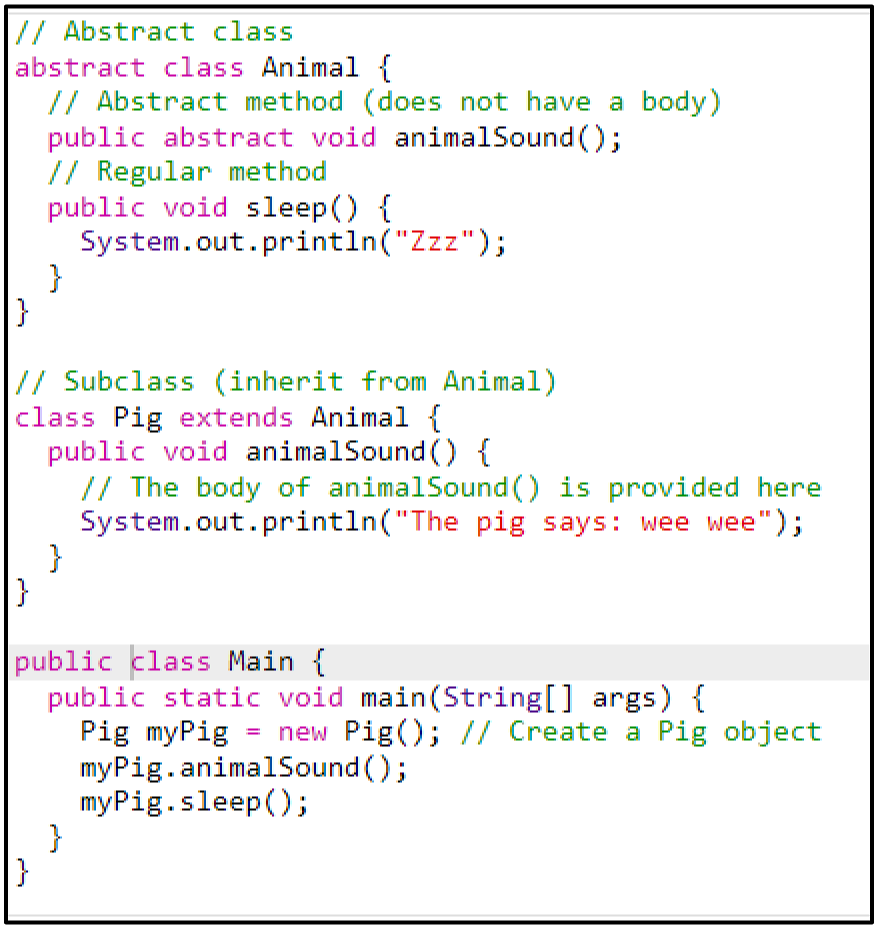
An abstract class can have both abstract and regular methods:

```java
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

From the example above, it is not possible to create an object of the Animal class:

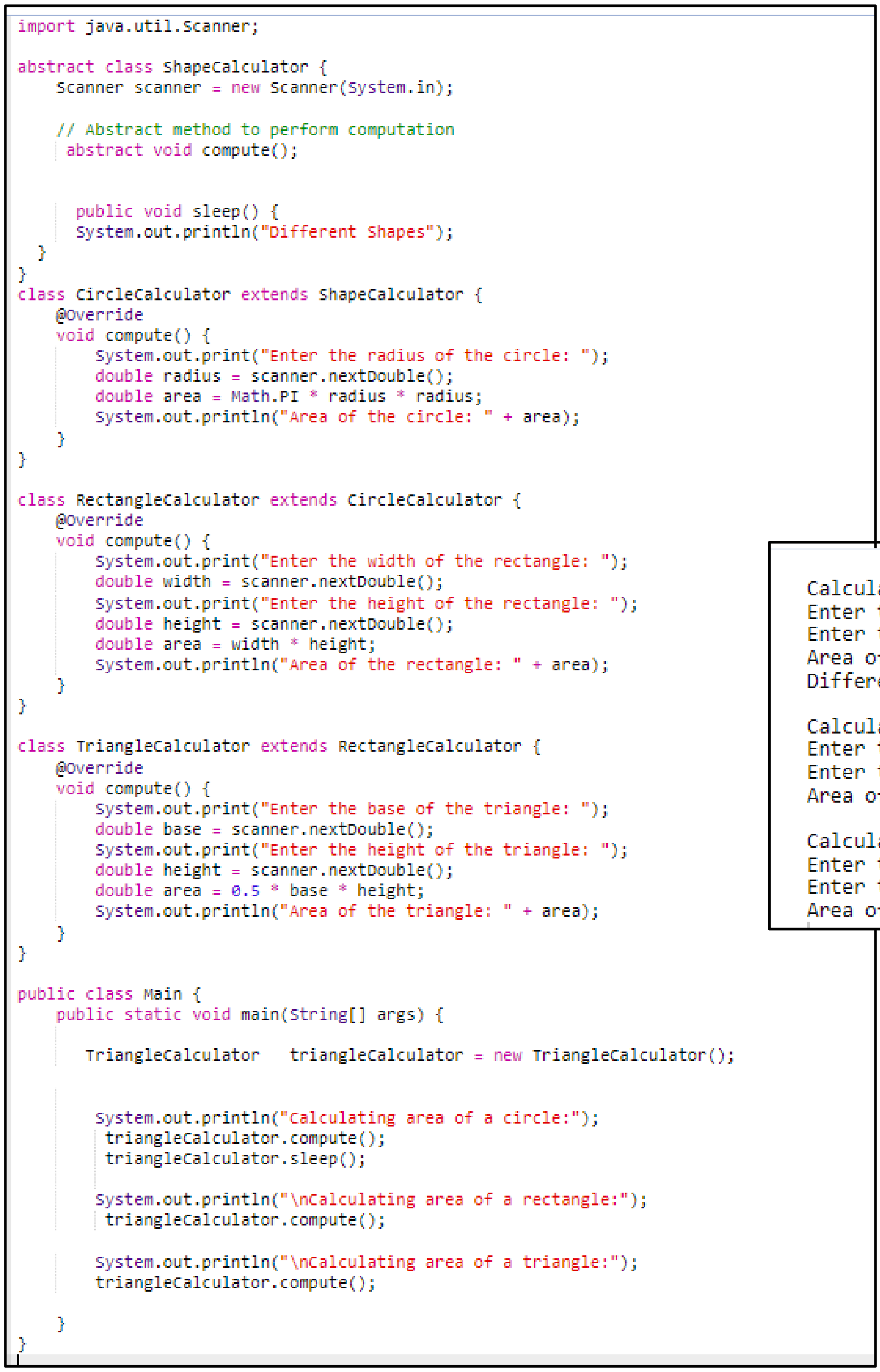
```java
Animal myObj = new Animal(); // will generate an error
```

To access the abstract class, it must be inherited from another class. Let's convert the Animal class we used in the Polymorphism chapter to an abstract class:



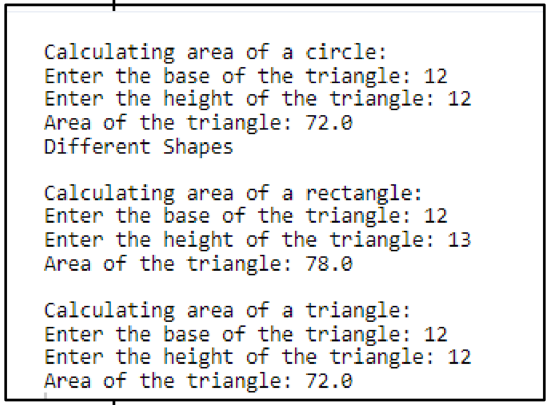
```java
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

public class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

ABSTRACTION EXAMPLE USING USER INPUT



```java
import java.util.Scanner;

abstract class ShapeCalculator {
    Scanner scanner = new Scanner(System.in);

    // Abstract method to perform computation
     abstract void compute();


      public void sleep() {
      System.out.println("Different Shapes");
  }
}
class CircleCalculator extends ShapeCalculator {
    @Override
    void compute() {
        System.out.print("Enter the radius of the circle: ");
        double radius = scanner.nextDouble();
        double area = Math.PI * radius * radius;
        System.out.println("Area of the circle: " + area);
    }
}

class RectangleCalculator extends CircleCalculator {
    @Override
    void compute() {
        System.out.print("Enter the width of the rectangle: ");
        double width = scanner.nextDouble();
        System.out.print("Enter the height of the rectangle: ");
        double height = scanner.nextDouble();
        double area = width * height;
        System.out.println("Area of the rectangle: " + area);
    }
}

class TriangleCalculator extends RectangleCalculator {
    @Override
    void compute() {
        System.out.print("Enter the base of the triangle: ");
        double base = scanner.nextDouble();
        System.out.print("Enter the height of the triangle: ");
        double height = scanner.nextDouble();
        double area = 0.5 * base * height;
        System.out.println("Area of the triangle: " + area);
    }
}

public class Main {
    public static void main(String[] args) {

       TriangleCalculator   triangleCalculator = new TriangleCalculator();


        System.out.println("Calculating area of a circle:");
         triangleCalculator.compute();
         triangleCalculator.sleep();

        System.out.println("\nCalculating area of a rectangle:");
         triangleCalculator.compute();

        System.out.println("\nCalculating area of a triangle:");
        triangleCalculator.compute();

    }
}
```

OUTPUT



```
Calculating area of a circle:
Enter the base of the triangle: 12
Enter the height of the triangle: 12
Area of the triangle: 72.0
Different Shapes

Calculating area of a rectangle:
Enter the base of the triangle: 12
Enter the height of the triangle: 13
Area of the triangle: 78.0

Calculating area of a triangle:
Enter the base of the triangle: 12
Enter the height of the triangle: 12
Area of the triangle: 72.0
```

Java Interface

Interfaces

Another way to achieve abstraction in Java, is with interfaces.

An interface is a completely "**abstract class**" that is used to group related methods with empty bodies:

```
// interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
```

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class:

```
interface Animal {
  public  void animalSound(); // interface method (does not have a body)
 public   void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
 public  void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
   public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}

public class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Notes on Interfaces:

- Like **abstract classes**, interfaces **cannot** be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default abstract and public
- Interface attributes are by default public, static and final
- An interface cannot contain a constructor (as it cannot be used to create objects)

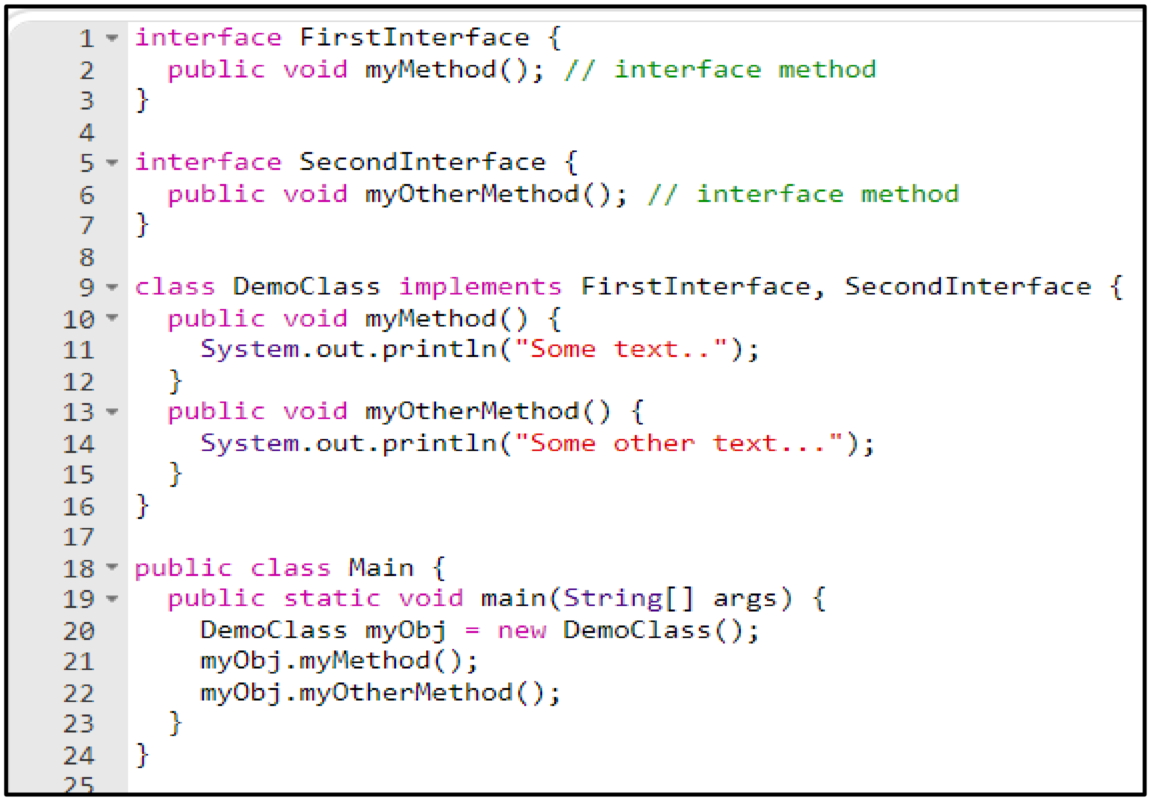
Why And When To Use Interfaces?

1) To achieve security - hide certain details and only show the important details of an object (interface).

2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces. **Note:** To implement multiple interfaces, separate them with a comma (see example below).

Multiple Interfaces

To implement multiple interfaces, separate them with a comma:



```java
interface FirstInterface {
  public void myMethod(); // interface method
}

interface SecondInterface {
  public void myOtherMethod(); // interface method
}

class DemoClass implements FirstInterface, SecondInterface {
  public void myMethod() {
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}

public class Main {
  public static void main(String[] args) {
    DemoClass myObj = new DemoClass();
    myObj.myMethod();
    myObj.myOtherMethod();
  }
}
```