More About
Derived Classes and Inheritance

Chapter 9

The Principle of Encapsulation

- Normally we don't want instance variable of a class to be accessible to methods in other classes.
- The *access modifier* **`private`** makes an instance variable or method inaccessible to methods in other classes.
  - If we want other classes to be able to *see* the value of an instance variable we can provide a public accessor method.
- What about derived classes?
  - A private member is inaccessible even to methods in derived classes.

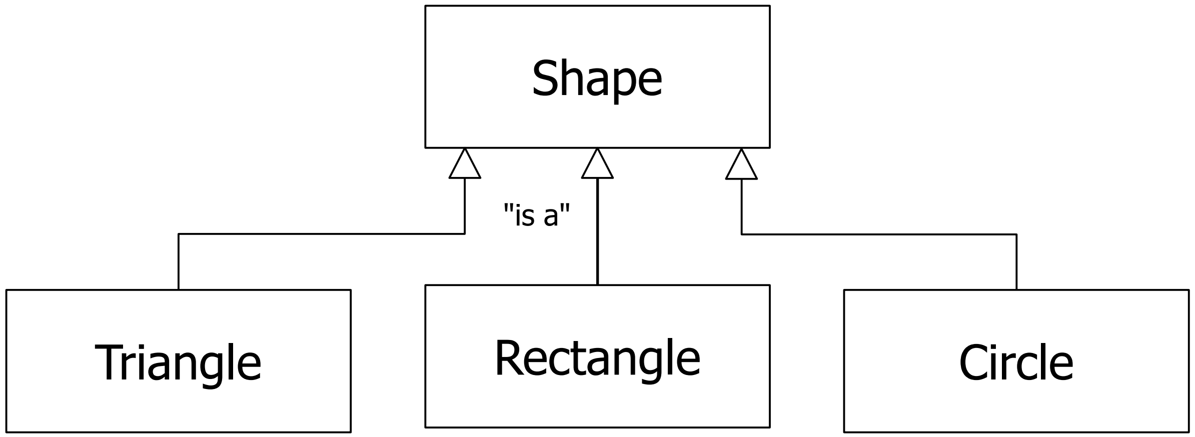
Access Modifiers in Derived Classes

- Derived classes have a special relationship to their base classes.
  - The base class is effectively a part of the derived class.

- We typically want a derived class to have access to members of the base class that are not accessible to the general public.

- We can do this with the *protected* access modifier.

Recall our hierarchy of *shapes* from last class.



Another Shape

- Suppose now that we need a class Square.

- Clearly a square "is a" rectangle.
  - A square is a special case of a rectangle.
  - Length = Width.

- We can use a square wherever we need a rectangle.

- Class Square should be a subclass of class Rectangle.

Class Square

- Class Square should be a subclass of class Rectangle.

- But it doesn't have any new members.
- It has a *restriction*.
  - Length must equal width.

Download Shape files from the class web site

http://www.cse.usf.edu/~turnerr/Programming_Concepts/Downloads/2016_04_25_Shapes/

We will add Square.java

Implementing Class Square

Look at Rectangle.java

```
//-----------------------------------------------------
   // Constructor - Initializes instance variables
   //----------------------------------------------------
   public Rectangle(double length, double width)
   {
       super("Rectangle", 4, length*width);
       this.name = "Rectangle";
       this.length = length;
       this.width = width;
   }
```

If derived class Square invokes this constructor, the name will be set to Rectangle, not Square.

New Rectangle Constructor

- In order to support the derived class Square, we need a new constructor in class Rectangle.
  - *Overload* the constructor.

- Let the caller specify the name.

New Rectangle Constructor

```
//------------------------------------------
// Constructor for derived classes
//------------------------------------------
protected Rectangle(String name, double length, double width)
{
    super(name, 4, length*width);
    this.length = length;
    this.width = width;
}
```

- *protected* access modifier means that this constructor can only be invoked from a derived class.

Class Square

- Create a new file for class Square.
  - Square.java

- Content on next slide.

```
//*********************************************
//   Square.java
//
//   Represents a geometrical square
//
//*********************************************
public class Square extends Rectangle
{
    //----------------------------------------------
    // Constructor
    //----------------------------------------------
    public Square(double side)
    {
        super("Square", side, side);
    }
```

Uses the new overloaded constructor

Inherits toString method from class Rectangle

```
}
```

Add Test Code for Square

```
if (shapeName.equals("Rectangle"))
           {
               double length, width;
               System.out.print ("Length: ");
               length = keyboardScanner.nextDouble();
               System.out.print ("Width: ");
               width = keyboardScanner.nextDouble();
               System.out.println();

               // Create a Rectangle object
               new_shape = new Rectangle(length, width);
           }
           else if (shapeName.equals("Square"))
           {
               double side;
               System.out.print("Side: ");
               side = keyboardScanner.nextDouble();
               new_shape = new Square(side);
               System.out.println();
           }
```

Comment out test code for Triangle

```
else if (shapeName.equals("Triangle"))
            {
                // double base, height;
                // System.out.print("Base: ");
                // base  = keyboardScanner.nextDouble();
                // System.out.print ("Height: ");
                // height = keyboardScanner.nextDouble();
                // new_shape = new Triangle(base, height);
                // System.out.println();
            }
```

Shape Objects

- It really doesn't make sense to instantiate a generic Shape object.
  - Only the derived classes represent *real* shapes.

- The only purpose of class Shape is to server as a base class for the derived classes.
  - A single home for everthing that is common to all classes.

- Java has provision for classes like this:
  - *Abstract* class

Abstract Class

Textbook, page 461

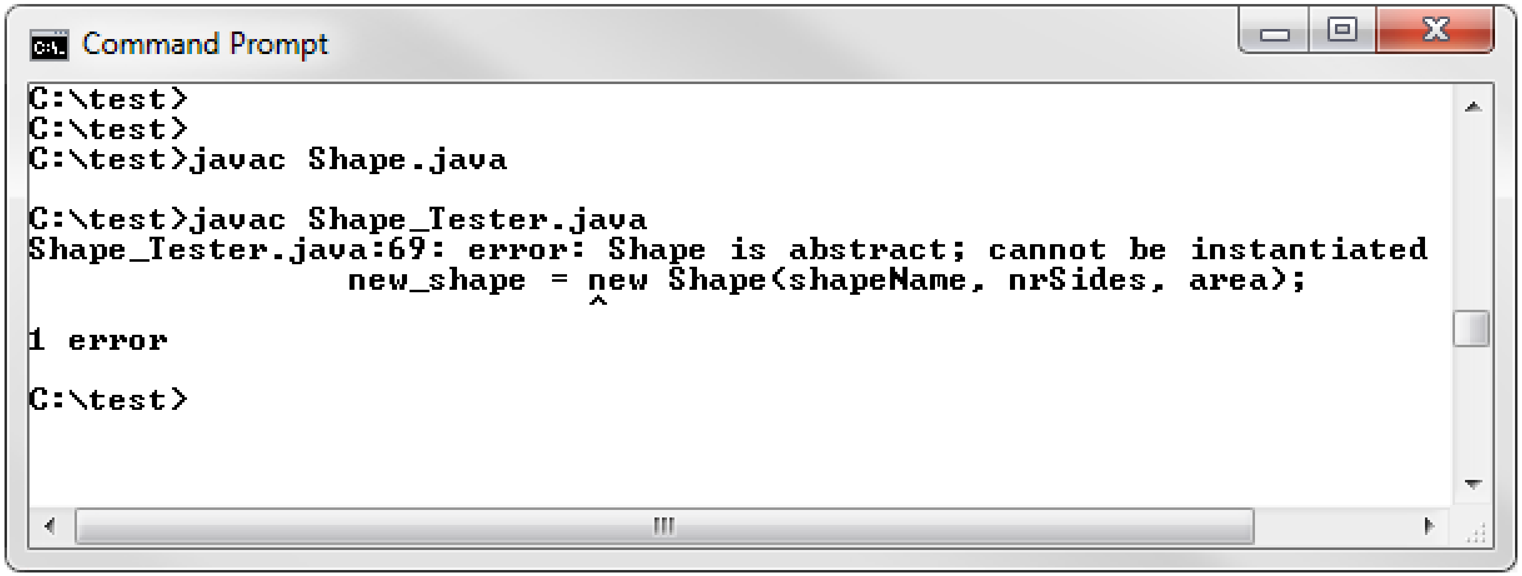
Key Concept

- The modifier *abstract* in the class header tells the compiler that this is to be an abstract class.
  - Cannot be instantiated.
  - Exists only to be a base class for derived classes.

- Let's make the Shape class abstract.

```
//***********************************************
//  Shape.java
//
//  Represents a geometrical shape
//
//***********************************************
public abstract class Shape
{
    private static int nr_shape_objects = 0;

    // Instance variables
    private String name;
    private int id;
    private int nr_sides;
    private double area;

    ...
```

Recompile and Test

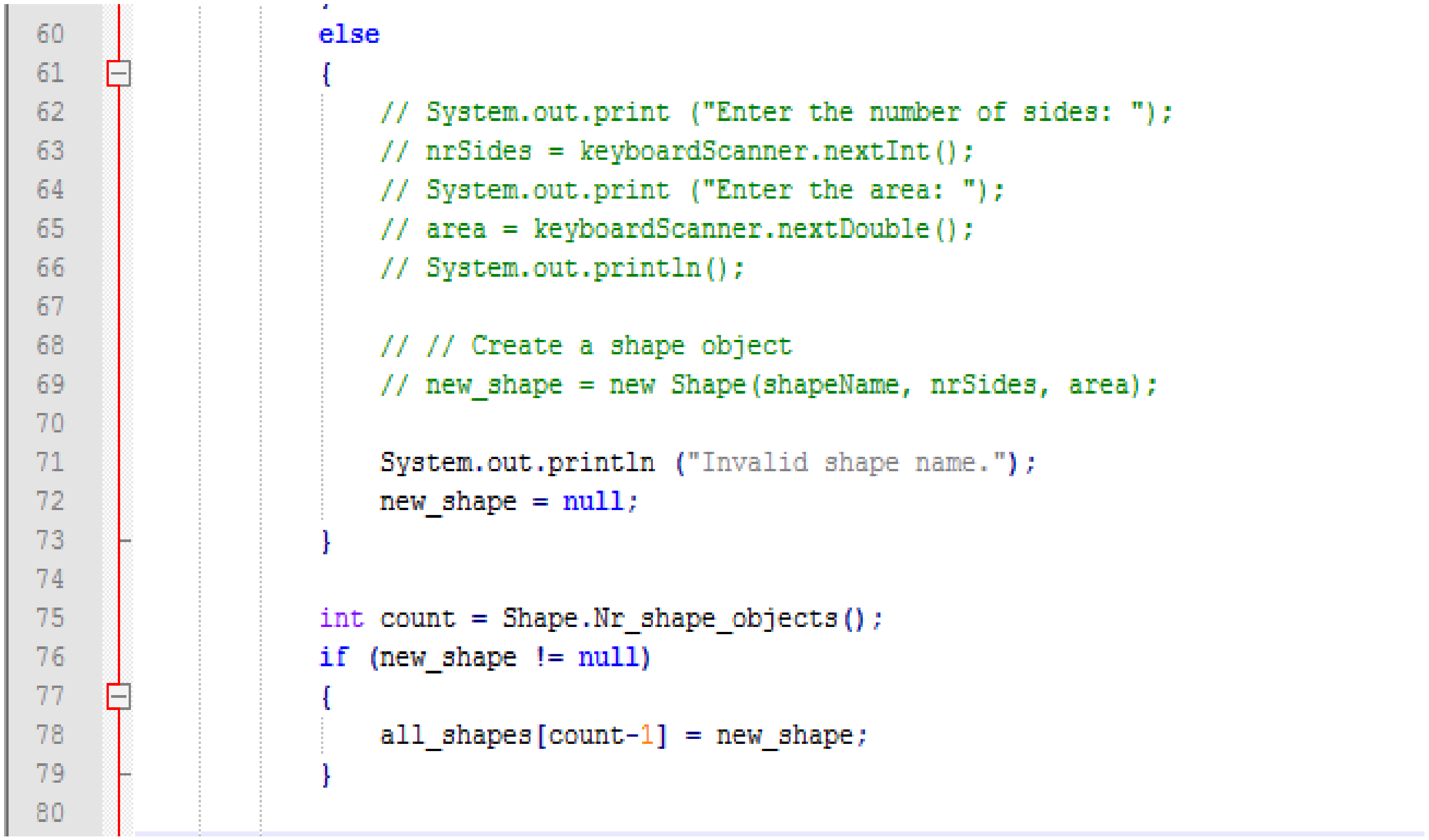


```
C:\test>
C:\test>
C:\test>javac Shape.java

C:\test>javac Shape_Tester.java
Shape_Tester.java:69: error: Shape is abstract; cannot be instantiated
                new_shape = new Shape(shapeName, nrSides, area);
                            ^
1 error

C:\test>
```

We have to revise Shape_Tester.java now.

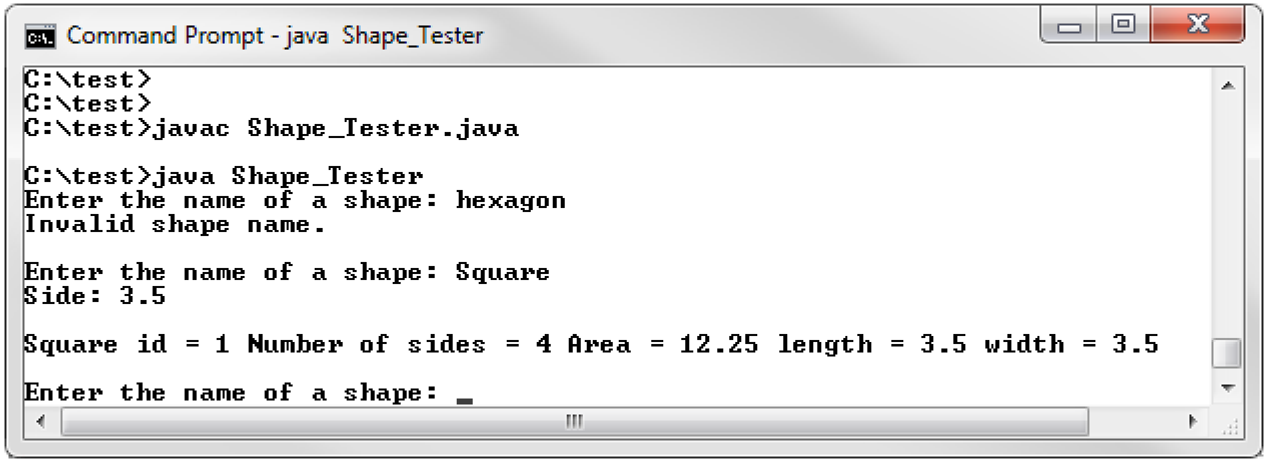
It can no longer instantiate generic Shape objects.

Update Shape_Test.java



```
        else
        {
            // System.out.print ("Enter the number of sides: ");
            // nrSides = keyboardScanner.nextInt();
            // System.out.print ("Enter the area: ");
            // area = keyboardScanner.nextDouble();
            // System.out.println();

            // // Create a shape object
            // new_shape = new Shape(shapeName, nrSides, area);

            System.out.println ("Invalid shape name.");
            new_shape = null;
        }

        int count = Shape.Nr_shape_objects();
        if (new_shape != null)
        {
            all_shapes[count-1] = new_shape;
        }

```

Revised Shape_Tester



```
C:\test>
C:\test>
C:\test>javac Shape_Tester.java

C:\test>java Shape_Tester
Enter the name of a shape: hexagon
Invalid shape name.

Enter the name of a shape: Square
Side: 3.5

Square id = 1 Number of sides = 4 Area = 12.25 length = 3.5 width = 3.5

Enter the name of a shape: _
```

Abstract Methods

- An abstract class can include abstract *methods*.
  - Methods defined only to be overridden in derived classes.

- Why do this?
  - It guarantees that any object of a derived class will have an implementation of that method.

  - A method in another class with a reference to an object of the base class can invoke the method without concern for which derived class it is.
    - Polymorphism!

Abstract Methods

- How do we make a method abstract?

- Let the class definition have a declaration for the method, but no definition.
- Include the keyword abstract in the method declaration.
- Example:
  - `abstract double perimeter();`

Abstract Methods

- If a an abstract base class includes an abstract method, every derived class must provide an implementation of the method.
  - Or else be an abstract class itself.

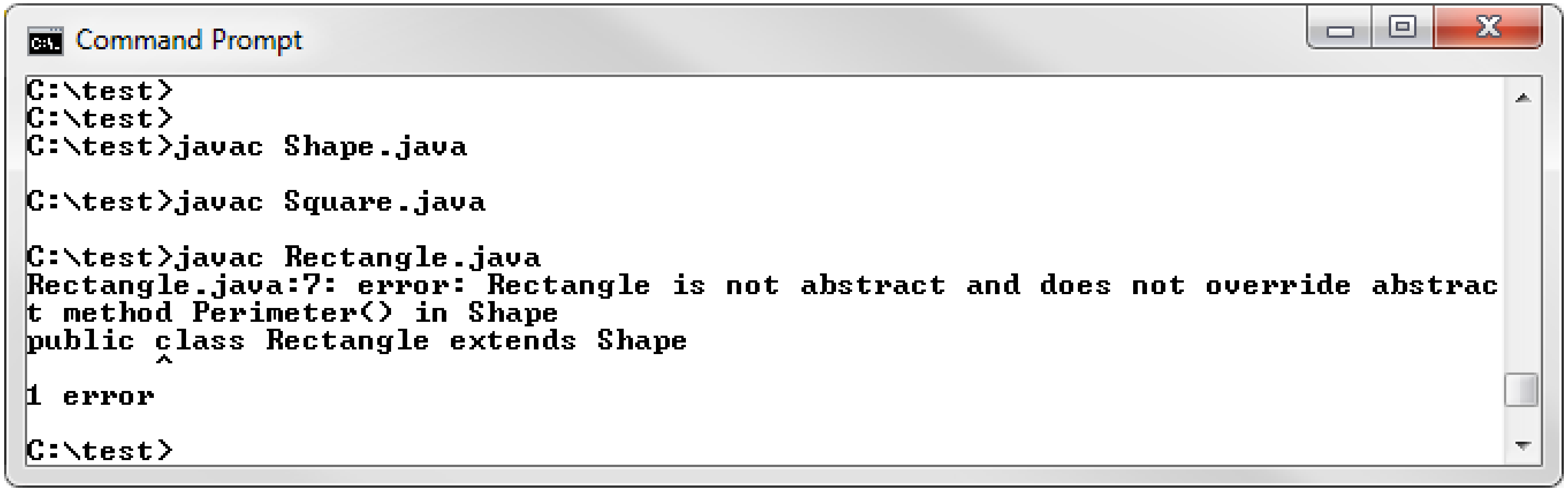
- Every *object* of a derived class will have the method.

Abstract Methods

Let's add an abstract Perimeter method to class shape.

```
//***************************************************
//   Shape.java
//
//   Represents a geometrical shape
//
//***************************************************
public abstract class Shape
{
    ...

    //-------------------------------------------------------
    // Returns a the perimeter of the Shape
    //-------------------------------------------------------
    abstract double Perimeter();

    ...
```

Compiling



```
C:\test>
C:\test>
C:\test>javac Shape.java

C:\test>javac Square.java

C:\test>javac Rectangle.java
Rectangle.java:7: error: Rectangle is not abstract and does not override abstrac
t method Perimeter() in Shape
public class Rectangle extends Shape
       ^
1 error

C:\test>
```

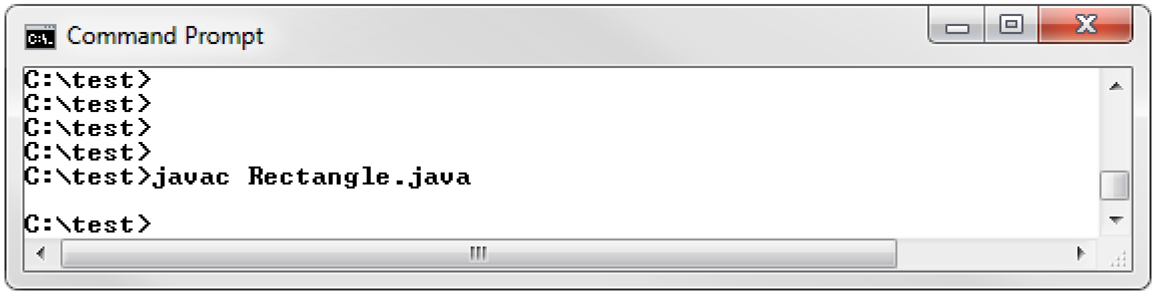
We must implement the Perimeter method in class Rectangle.

Class Rectangle

Add to class Rectangle:

```
public double Perimeter()
{
    return 2.0*length + 2.0*width;
}
```

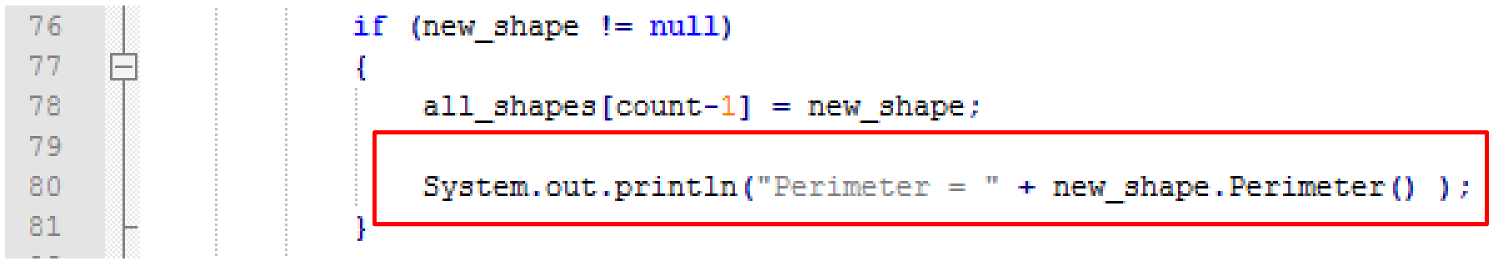
Now it compiles again.



```
C:\test>
C:\test>
C:\test>
C:\test>
C:\test>javac Rectangle.java

C:\test>
```

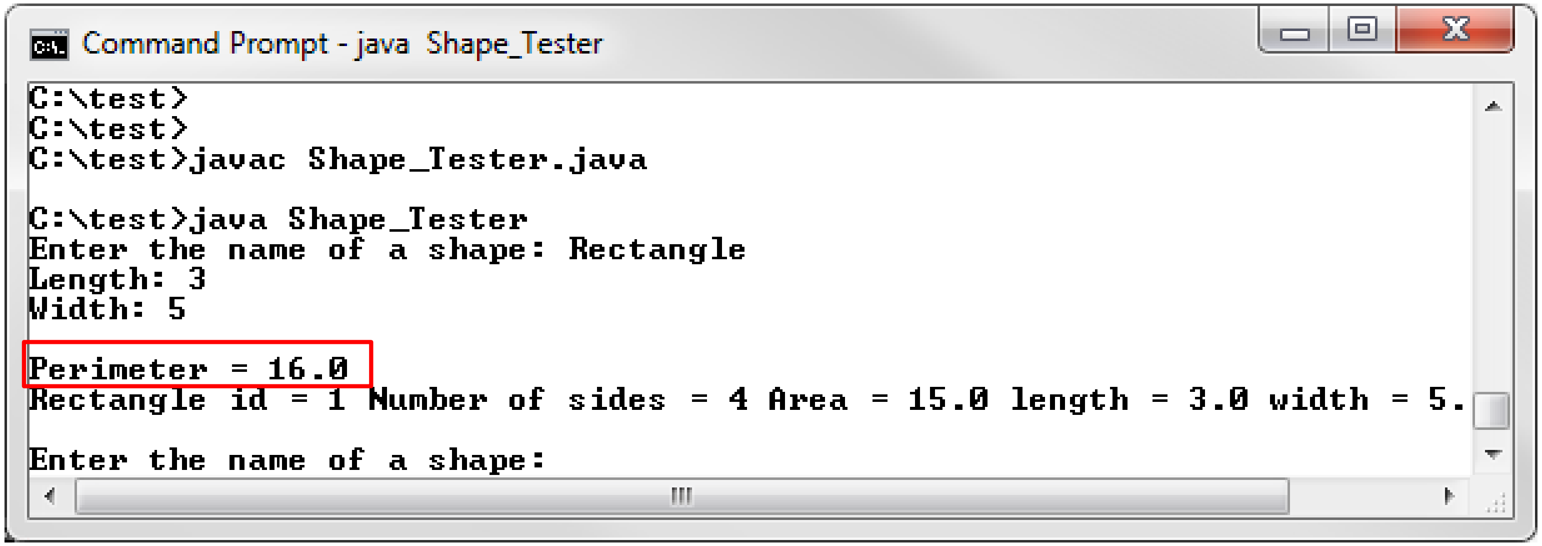
Add test code

Add to Shape_Tester.java:



```
            if (new_shape != null)
            {
                all_shapes[count-1] = new_shape;

                System.out.println("Perimeter = " + new_shape.Perimeter() );
            }
```

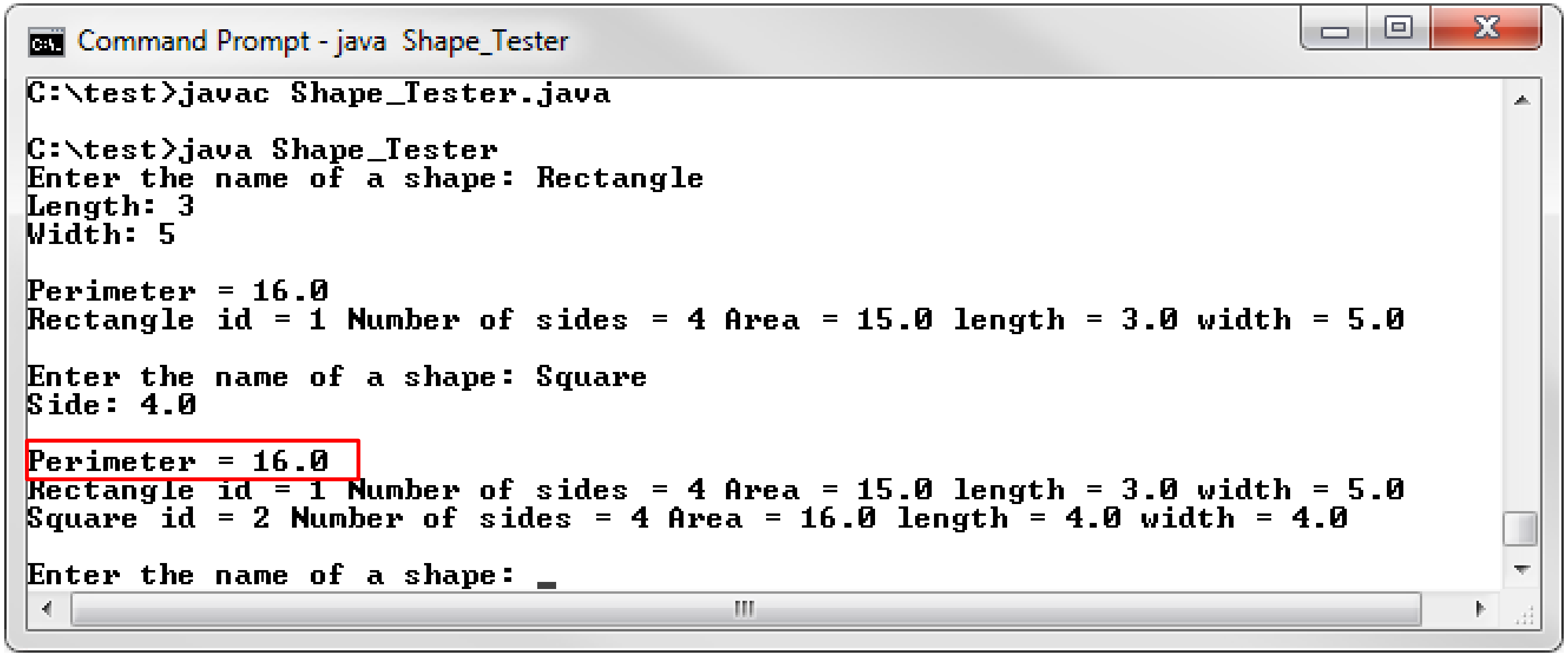
Testing



```
C:\test>
C:\test>
C:\test>javac Shape_Tester.java

C:\test>java Shape_Tester
Enter the name of a shape: Rectangle
Length: 3
Width: 5

Perimeter = 16.0
Rectangle id = 1 Number of sides = 4 Area = 15.0 length = 3.0 width = 5.

Enter the name of a shape:
```

What about Square?

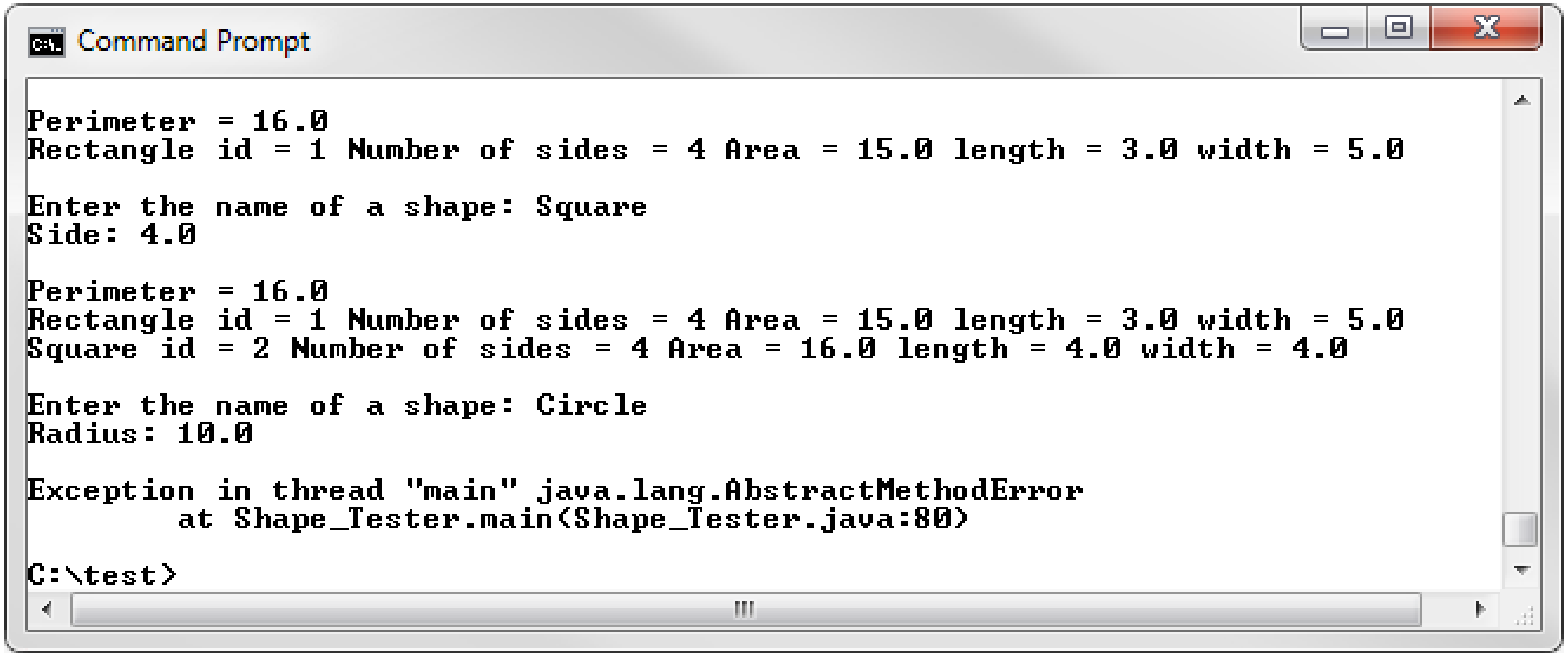
Testing Class Square



```
C:\test>javac Shape_Tester.java

C:\test>java Shape_Tester
Enter the name of a shape: Rectangle
Length: 3
Width: 5

Perimeter = 16.0
Rectangle id = 1 Number of sides = 4 Area = 15.0 length = 3.0 width = 5.0

Enter the name of a shape: Square
Side: 4.0

Perimeter = 16.0
Rectangle id = 1 Number of sides = 4 Area = 15.0 length = 3.0 width = 5.0
Square id = 2 Number of sides = 4 Area = 16.0 length = 4.0 width = 4.0

Enter the name of a shape: _
```

Class Square *inherits* the Perimeter method from its base class, Rectangle.

Testing Class Circle



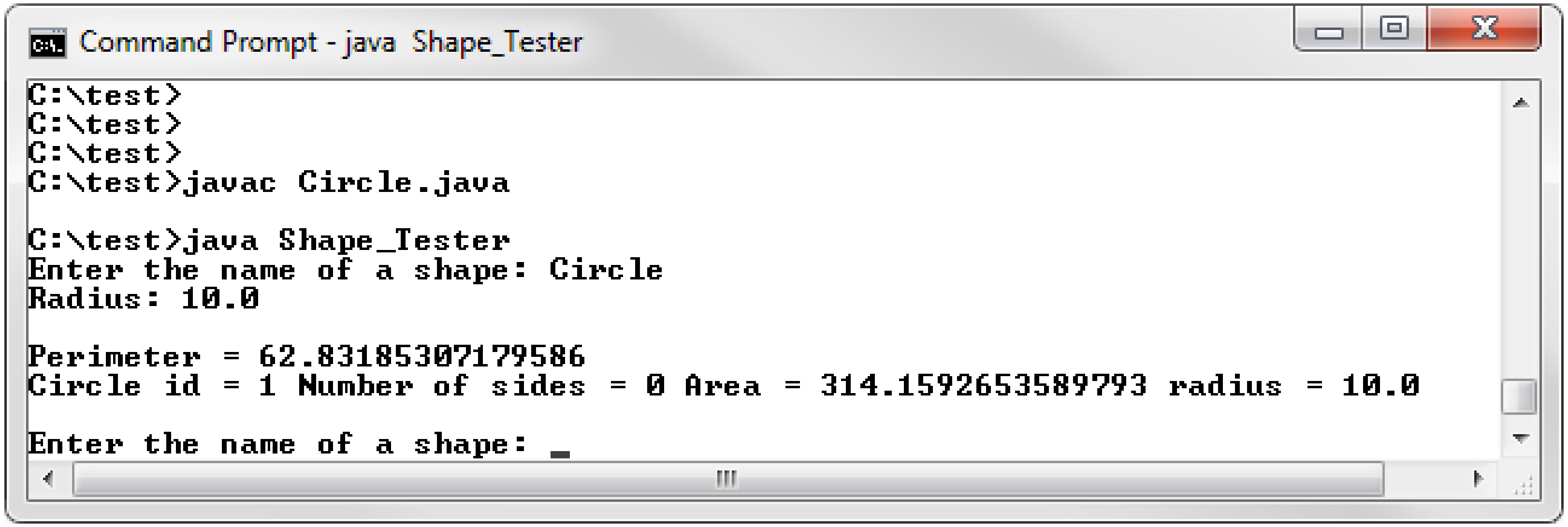
```
Perimeter = 16.0
Rectangle id = 1 Number of sides = 4 Area = 15.0 length = 3.0 width = 5.0

Enter the name of a shape: Square
Side: 4.0

Perimeter = 16.0
Rectangle id = 1 Number of sides = 4 Area = 15.0 length = 3.0 width = 5.0
Square id = 2 Number of sides = 4 Area = 16.0 length = 4.0 width = 4.0

Enter the name of a shape: Circle
Radius: 10.0

Exception in thread "main" java.lang.AbstractMethodError
        at Shape_Tester.main(Shape_Tester.java:80)

C:\test>
```

Class Circle doesn't inherit the Perimeter method, and does not implement it.

Add to Class Circle

```
public double Perimeter()
{
    return 2.0*Math.PI*radius;
}
```

Perimeter of a Circle



```
C:\test>
C:\test>
C:\test>
C:\test>javac Circle.java

C:\test>java Shape_Tester
Enter the name of a shape: Circle
Radius: 10.0

Perimeter = 62.83185307179586
Circle id = 1 Number of sides = 0 Area = 314.1592653589793 radius = 10.0

Enter the name of a shape: _
```

Summary

- An abstract class can define abstract methods.
  - No definition, just a declaration.
- A derived class must provide a definition
  - unless it is also abstract.
- Every *object* of a class derived from the abstract class will have the abstract methods.