# Computer Science II
# CS 1101 – Introduction to Programming

# Objects in Java – example

```
public class Rectangle {

        private double width;
        private double length;

        public Rectangle()
        { }


        public void setWidth(double w)
        {
                width = w;
        }

        public void setLength(double l)
        {
                length = l;
        }

        public double getLength()
        {
                return length;
        }

        public double getWidth()
        {
                return width;
        }
}
```

# Objects in Java - review

- We have a class definition for a Rectangle – every Object created using this outline will have:
  - Two instance variables:
    ```
    private double width
    private double length
    ```
  - One constructor:
    ```
    public Rectangle()
    ```
  - Two set methods:
    ```
    public void setLength(double w)
    public void setWidth(double w)
    ```
  - Two get methods:
    ```
    public double getLength()
    public double getWidth()
    ```

# Objects in Java – instance variables

```
private double width
private double length
```

- Instance variables are shared between all methods within a class file, but *may* specify how they are shared to other methods outside the class!
  - Specifying **`private`** makes sure the variable can only be accessed by methods/statements within class **`Rectangle`**
  - This gives us control over the variables' contents in our design

# Objects in Java – constructors

```
public Rectangle()
```

- Constructors are methods without a specified return type that share their name with the class name
  - Constructors are called whenever a new **`Rectangle`** Object is created using this class file

# Objects in Java – set/mutator methods

```
public void setLength(double l)
public void setWidth(double w)
```

- Set methods are regular methods with a specific (and common) purpose: to modify a (typically private) instance variable
  - Set methods typically assign a parameter value to an instance variable, but can do other work too
    (e.g., count the number of times a variable has been modified)
  - Specifying `public` makes sure these methods can be called from outside the class file

# Objects in Java – get/accessor methods

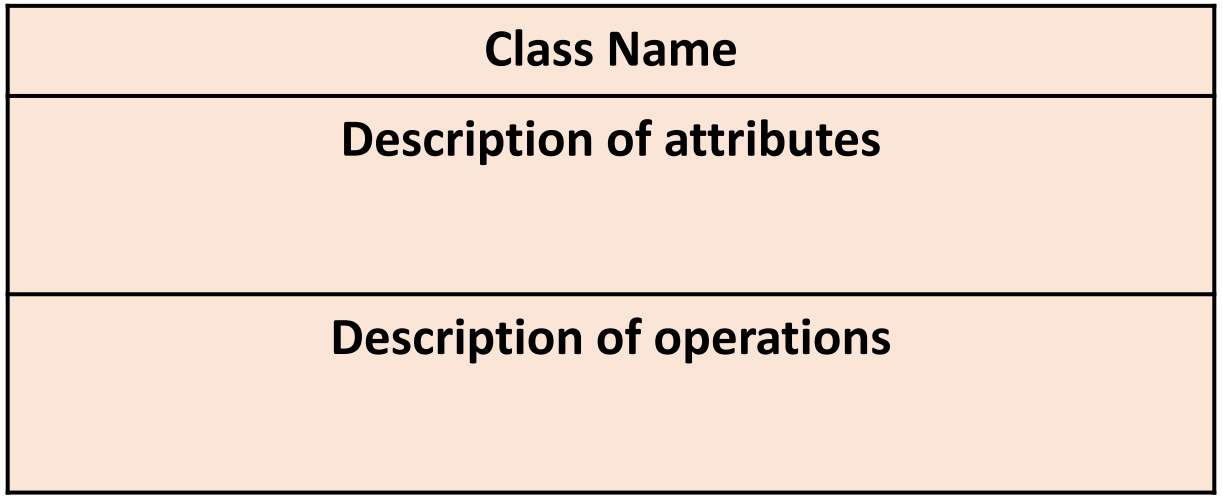
```
public double getLength()
public double getWidth()
```

- Get methods are regular methods with a specific (and common) purpose: to return the value of a (typically private) instance variable
  - Get methods can do other work too
    (e.g., count the number of times a variable has had its value retrieved)
  - Specifying `public` makes sure these methods can be called from outside the class file

# Objects in Java – aiding the design process

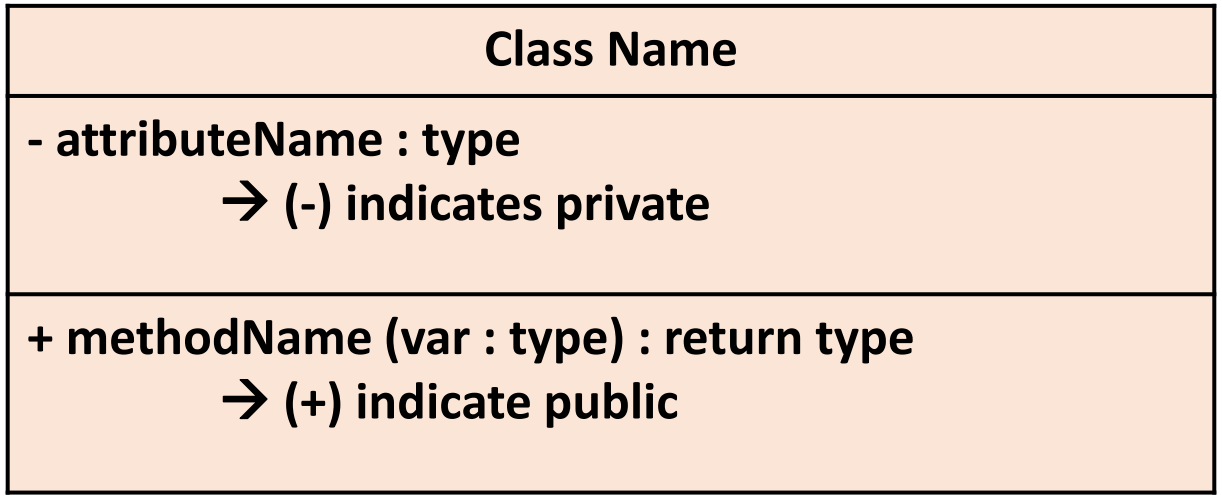
- Designing a class can sometimes take as much time and effort as actually writing its code

- To help with this process, we will sometimes use UML (Unified Modeling Language) diagrams to represent a class definition

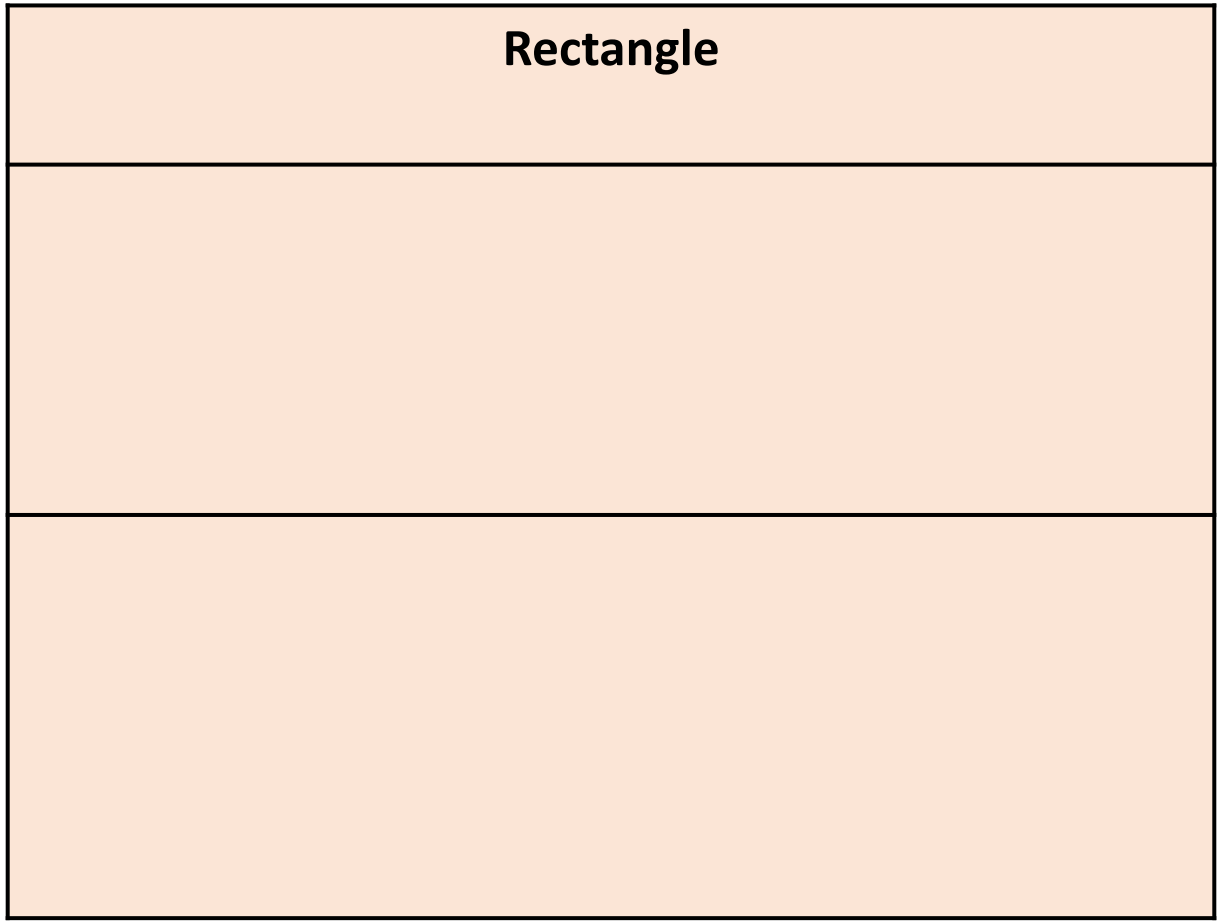


# Objects in Java – aiding the design process

- List attributes (variables) by giving names and types, prefixed with +/- to indicate public/private
- List operations (methods) by giving names, parameters, and return type, prefixed with +/- to indicate public/private



# Objects in Java – aiding the design process



# Objects in Java - designing

```
public double getArea() {



}
```

# Using your Objects

- All of this *defines* a class, but how can we actually use it?
- Say we want to use this Rectangle class to create a Rectangle object with dimensions 11 x 20, then print its area
- First, we write the class file as given, and compile this independently
- Then, we write another class file to create Rectangle Objects (like RectangleDemo.java)
  - For now, these class files can “see” each other, so long as they are in the same directory

# Using your Objects

```
public class RectangleDemo {
      public static void main(String[] args) {
            Rectangle rect = new Rectangle();




      }
}
```

```
> Area is 220.0
```

# Using your Objects

```
Rectangle rect = new Rectangle();
```

- This line performs the following:
  - Creates a variable named **`rect`** that will store a reference to an Object of type Rectangle
  - Creates a **`new`** Rectangle Object by calling the constructor method within the Rectangle class file
  - Stores a reference to the new Rectangle within the variable **`rect`**

# Using your Objects

```
Rectangle rect = new Rectangle();
```

- The Rectangle reference is stored on the *stack* within a local variable named **`rect`**

# Using your Objects

```
Rectangle rect = new Rectangle();
```

- The Object itself is stored on the *heap*, including its methods and its instance variables

- This Object is an *instance* of the class Rectangle (that is, it was created using the Rectangle class file blueprint)

# Using your Objects

- Methods within the Rectangle class file are called using a familiar format

```
<objectName>.<methodName>(<Parameters>)
```

- The `methodName` and `Parameters` refer to choices made within the class file that defines this type of Object
  (Rectangle.java)

- The `objectName` refers to a choice made in the code we are writing that uses the class file
  (RectangleDemo.java)

# Using your Objects

- Using this approach calls a **method** from the Rectangle class which operates on the Object referred to in **`rect`**

```
<objectName>.<methodName>(<Parameters>)
```

- This operates on the data (variables) stored within *this particular instance* of a Rectangle

# Ex: Instances of Objects

```
public class RectangleDemo {
        public static void main(String[] args) {
                Rectangle rect1 = new Rectangle();
                Rectangle rect2 = new Rectangle();









        }
}
```

```
> Area is 220.0
  Area is 15.0
```

# Object constructors

- Recall that constructors “look like” methods, but:
  - Have no return type
  - Always share their name with the class name

- They may take parameters, just like regular methods

- How can we use this to make a constructor that sets initial values for the length and width?

# Object constructors - Rectangle.java

```
public class Rectangle {
      private double width;
      private double length;

      public Rectangle() { }

      public Rectangle(double l, double w) {



      }

      // The other methods go down here...
}
```

# Object constructors – RectangleDemo.java

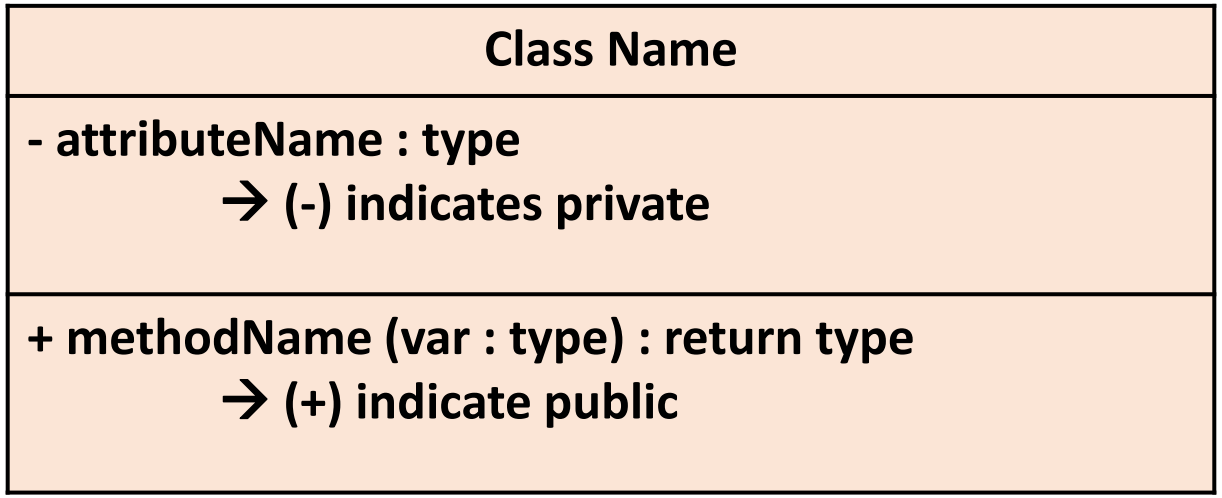
```
public class RectangleDemo {
      public static void main(String[] args) {
            Rectangle rect1 = new Rectangle(11, 20);
            Rectangle rect2 = new Rectangle(5, 3);

            System.out.println("Area is " + rect1.getArea());
            System.out.println("Area is " + rect2.getArea());

      }
}
```

```
> Area is 220.0
  Area is 15.0
```

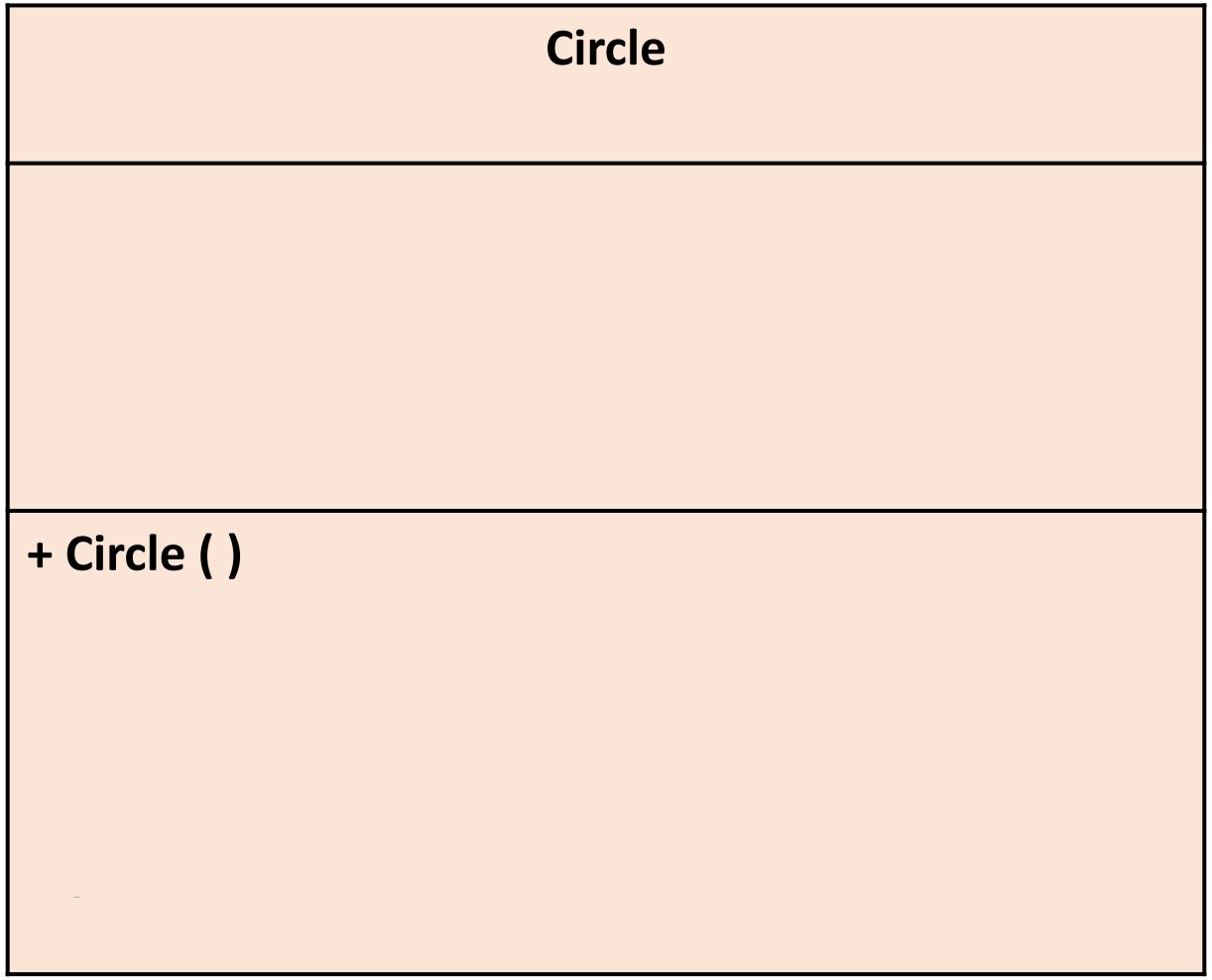
# Objects in Java – designing a new Object

Design a class definition for a Circle object.

- The object should be capable of being created with a **user-defined radius**
- The class definition should have methods to determine area and circumference



# Objects in Java – designing a new Object



# Objects in Java – designing a new Object

```
public class Circle {



        public Circle()
        { }









}
```

# Objects in Java – CircleDemo.java

```
import java.util.Scanner;
public class CircleDemo {
        public static void main(String[] args) {
                Circle circ1;
                double radius;
                Scanner kb = new Scanner(System.in);

                System.out.print("Enter the radius: ");
                radius = kb.nextDouble();

                circ1 = new Circle(radius);

                System.out.println("A = " + circ1.getArea());
                System.out.println("C = " + circ1.getCircumference());

        }
}
```

```
> Enter the radius: 5
```

# Objects in Java – using the toString method

- The toString method is a common method implemented by Objects in Java

- The idea is to give a String representation of the underlying Object
  - Exactly how this is done is implementation specific

- Writing your own toString method lets you decide what *really matters* about Objects belonging to your class
  - Otherwise, Java provides a default toString method that tends to print “garbage”

# Objects in Java – writing the toString method

- Within the Rectangle class file, the toString method could give the length and width

```
public String toString() {
      String ts;
      ts = "Rectangle: ";
      ts += " Length=" + length;
      ts += " Width=" + width;
      return ts;
}
```

# Objects in Java – writing the toString method

- Within the Circle class file, the toString method could give its radius

```
public String toString() {



}
```

# Objects in Java – writing the toString method

- Within the CircleDemo class file, the **`toString`** method can be called on any *instance* of the class

```
public class CircleDemo {
     public static void main(String[] args) {
          Circle circ1 = new Circle(4);
          Circle circ2 = new Circle(5);

          System.out.println(circ1.toString());
          System.out.println(circ2);
}
```

**Note that the `toString` method is called "by default" when Used in a print statement!**

# Objects in Java – problems with scope

```
public class Rectangle {    // What is wrong with this class?
      private double width;
      private double length;

      public Rectangle()
      { }

      public void setWidth(double width) {
            width = width;
      }

      public void setLength(double length) {
            length = length;
      }
```

# Objects in Java – problems with scope

- There are no compilation or syntax errors here!
  - The problem is one of design or logic

- When dealing with the *scope* of variables, Java always takes the “most local” variable with a matching name
  - This is true with loop variables as well

- But in this case: Java looks on the stack for local variables, not the heap

# Objects in Java – problems with scope

- To get around this problem, we can use the **`this`** keyword to refer to the *current instance* of a class
  - This forces Java to look for variables (of these names) on the *heap*, within the Object's attributes

```
public void setWidth(double width) {
        width = this.width;
}

public void setLength(double length) {
        length = this.length;
}
```