Middlesex University London

Lecture 11

OOP Thinking

THIS WEEK

- Abstarction
- Encapsulation
- Association
- Composition and Aggregation

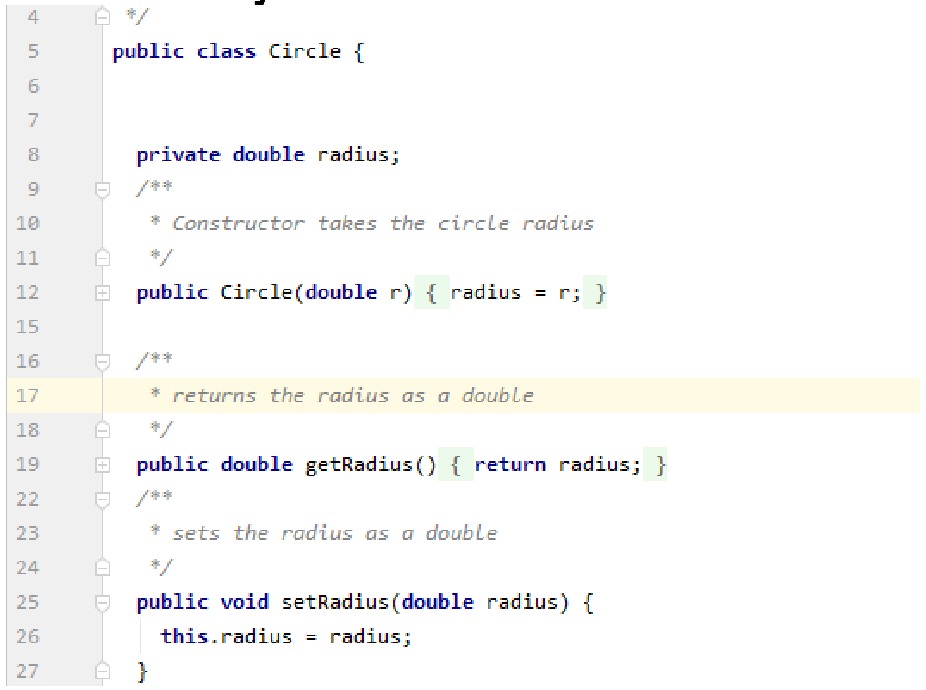
CLASS ABSTRACTION AND ENCAPSULATION

Class abstraction is separation of class implementation from the use of a class. The details of implementation are encapsulated and hidden from the user. This is known as class encapsulation.

- In Java, Abstraction is supported using interface and abstract class while Encapsulation is supported using access modifiers e.g. public, private and protected.

ENCAPSULATION

- Encapsulation can be achieved by declaring all the variables in the class as **private** and writing **public** methods in the class to set and get the values of variables. It basically creates a shield and the code cannot be accessed outside the shield or by

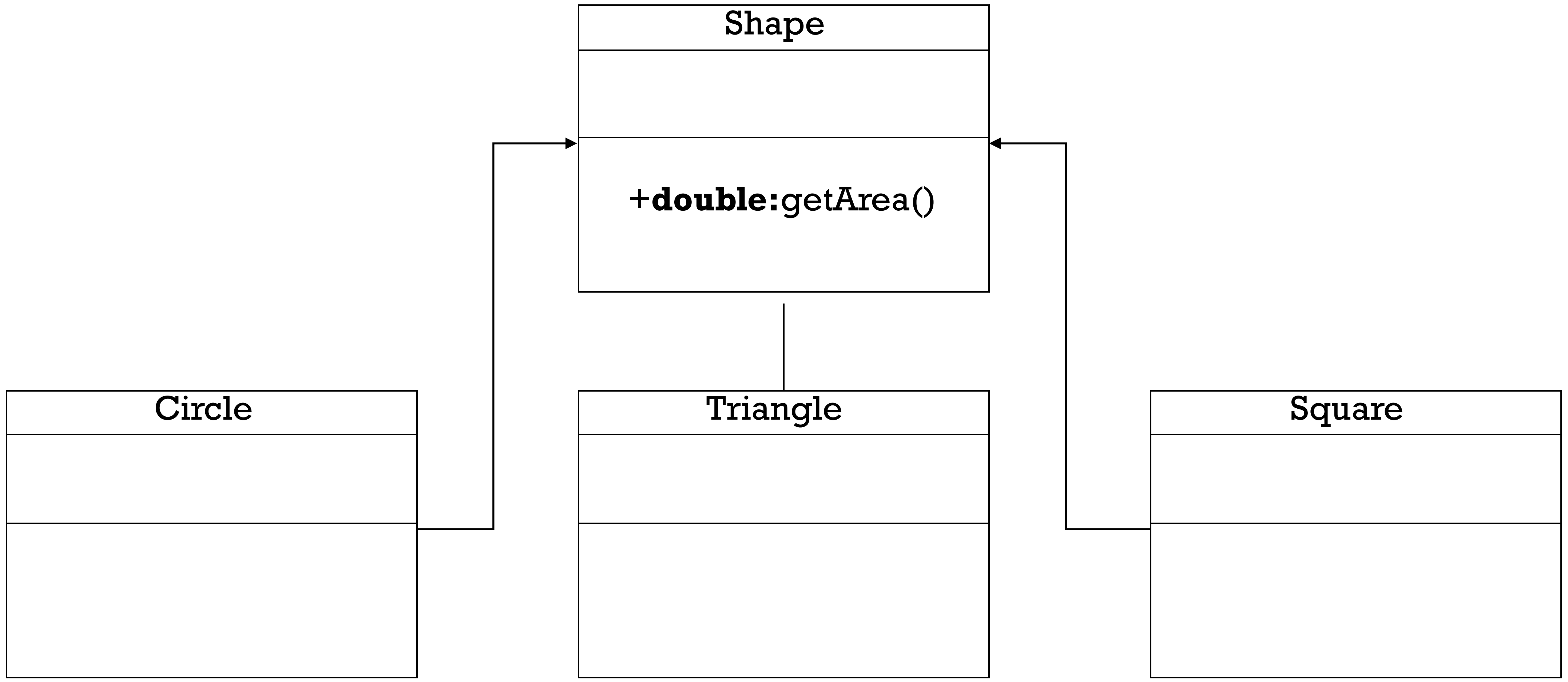


```java
 */
public class Circle {


  private double radius;
  /**
   * Constructor takes the circle radius
   */
  public Circle(double r) { radius = r; }

  /**
   * returns the radius as a double
   */
  public double getRadius() { return radius; }
  /**
   * sets the radius as a double
   */
  public void setRadius(double radius) {
    this.radius = radius;
  }
```

ABSTRACTION

- Abstract classes:
  - abstract classes can’t be instantiated, only subclassed.
  - other classes extend abstract classes.
  - can have both abstract and concrete methods.
  - similar to interfaces (we will be talking about it later), but (1) can implement methods, (2) fields can have various access modifiers, and (3) subclasses can only extend one abstract class.
- Abstract methods:
  - abstract method bodies must be empty (no curly braces)
  - subclasses must implement the abstract class’s abstract methods

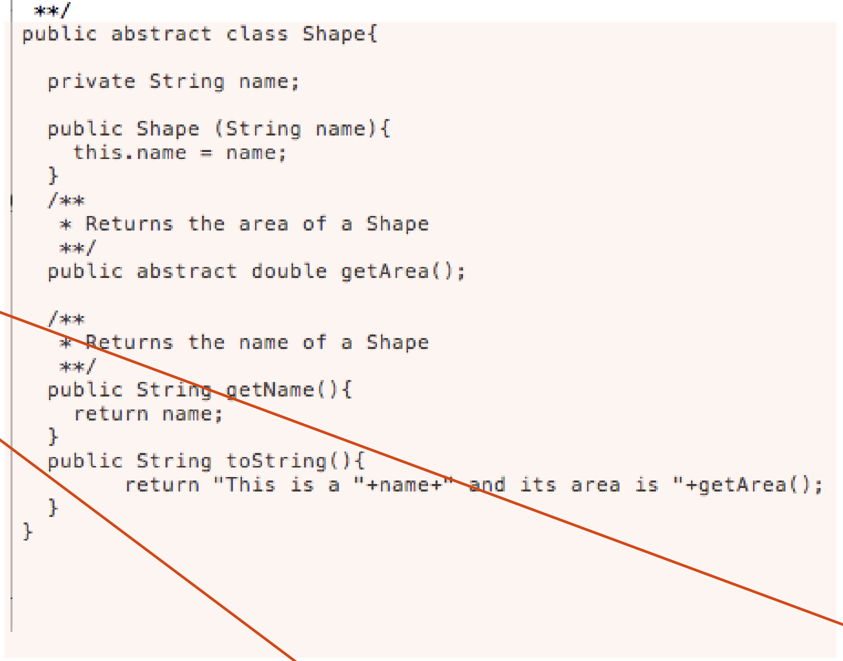
SHAPE HIERARCHY



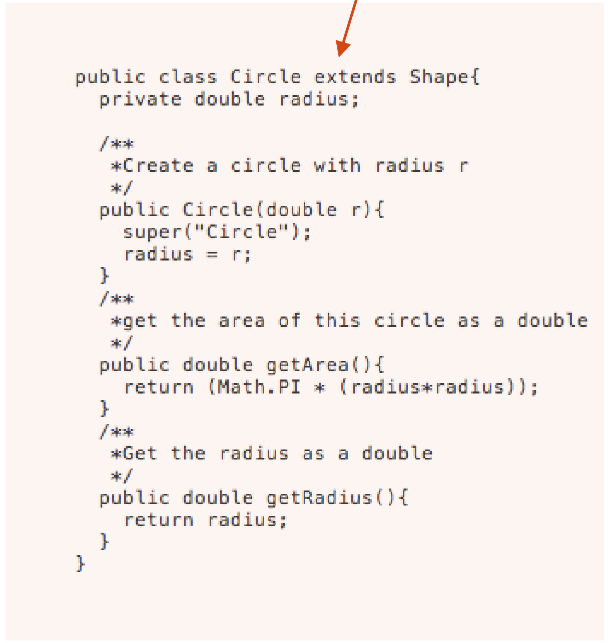
The extends keyword **extends** a class (indicates that a class is inherited from another class).

This is an **is-a** relationship. In this example, a Circle is a Shape, a Triangle is a Shape, a Rectangle is a Shape.

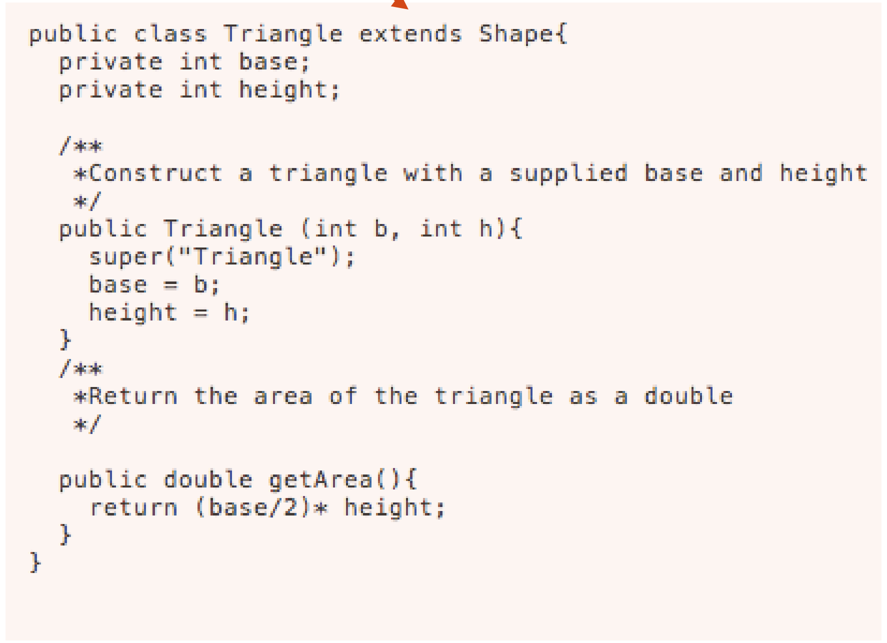
In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:
subclass (child) - the class that inherits from another class
superclass (parent) - the class being inherited from



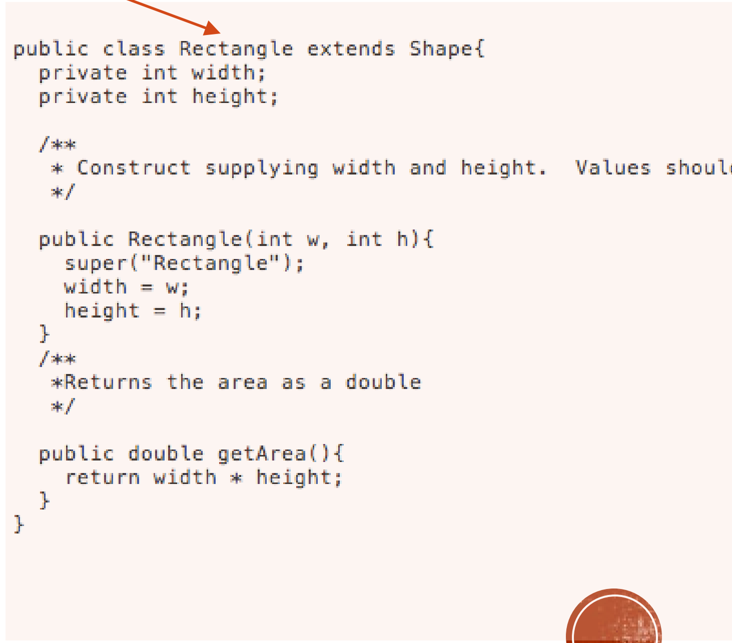
```
 **/
public abstract class Shape{

  private String name;

  public Shape (String name){
    this.name = name;
  }
  /**
   * Returns the area of a Shape
   **/
  public abstract double getArea();

  /**
   * Returns the name of a Shape
   **/
  public String getName(){
    return name;
  }
  public String toString(){
          return "This is a "+name+" and its area is "+getArea();
  }
}
```



```
public class Circle extends Shape{
  private double radius;

  /**
   *Create a circle with radius r
   */
  public Circle(double r){
    super("Circle");
    radius = r;
  }
  /**
   *get the area of this circle as a double
   */
  public double getArea(){
    return (Math.PI * (radius*radius));
  }
  /**
   *Get the radius as a double
   */
  public double getRadius(){
    return radius;
  }
}
```

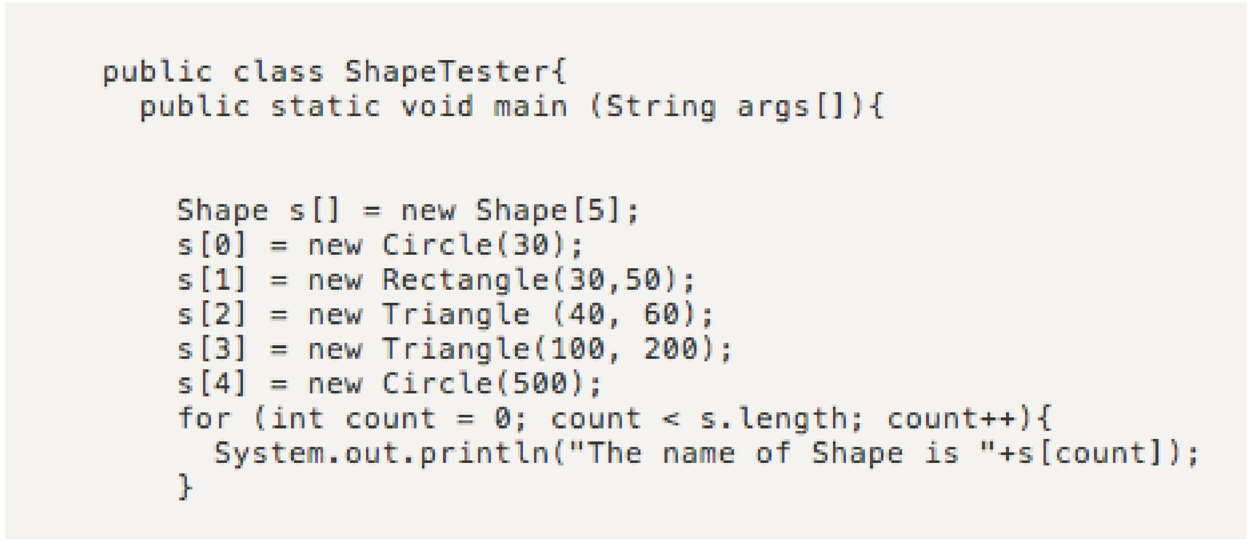


```
public class Triangle extends Shape{
  private int base;
  private int height;

  /**
   *Construct a triangle with a supplied base and height
   */
  public Triangle (int b, int h){
    super("Triangle");
    base = b;
    height = h;
  }
  /**
   *Return the area of the triangle as a double
   */

  public double getArea(){
    return (base/2)* height;
  }
}
```



```
public class Rectangle extends Shape{
  private int width;
  private int height;

  /**
   * Construct supplying width and height.  Values shoul
   */

  public Rectangle(int w, int h){
    super("Rectangle");
    width = w;
    height = h;
  }
  /**
   *Returns the area as a double
   */

  public double getArea(){
    return width * height;
  }
}
```

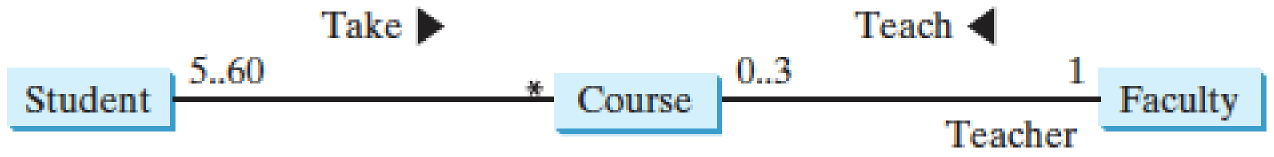
TESTER CLASS FOR SHAPES



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


    Shape s[] = new Shape[5];
    s[0] = new Circle(30);
    s[1] = new Rectangle(30,50);
    s[2] = new Triangle (40, 60);
    s[3] = new Triangle(100, 200);
    s[4] = new Circle(500);
    for (int count = 0; count < s.length; count++){
      System.out.println("The name of Shape is "+s[count]);
    }
```

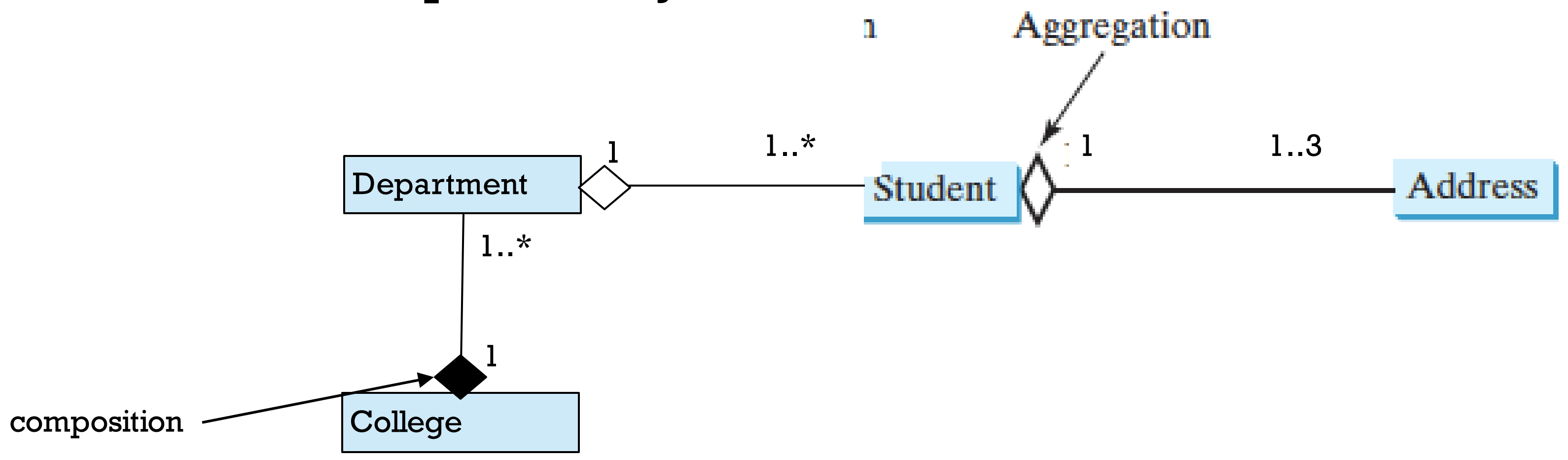
Association

Aggregation is a special form of association that represents an ownership relationship between two objects. Aggregation models **has-a** relationships. It is also used for code reusability in Java. In Java, a Has-A relationship simply means that an instance of one class has a reference to an instance of another class or an other instance of the same class.



Aggregation and Composition

- A relationship is a composition one when , the aggregated object cannot exist on its own. For example, "a student has a name" is a composition relationship between the Student class and the Name class because Name is dependent on Student, whereas "a student has an address" is an aggregation relationship between the Student class and the Address class because an address can exist by itself. Composition implies exclusive ownership. One object own



KEY THINGS YOU SHOULD UNDERSTAND AFTER WEEK 10

- Abstarction
- Encapsulation
- Association
- Composition and Aggregation

MASTERING YOUR SKILLS

- Read chapter 10 from the book and go over the exercises in the end of the chapter.