CISC 3115 TY2

Abstract Class and Method

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

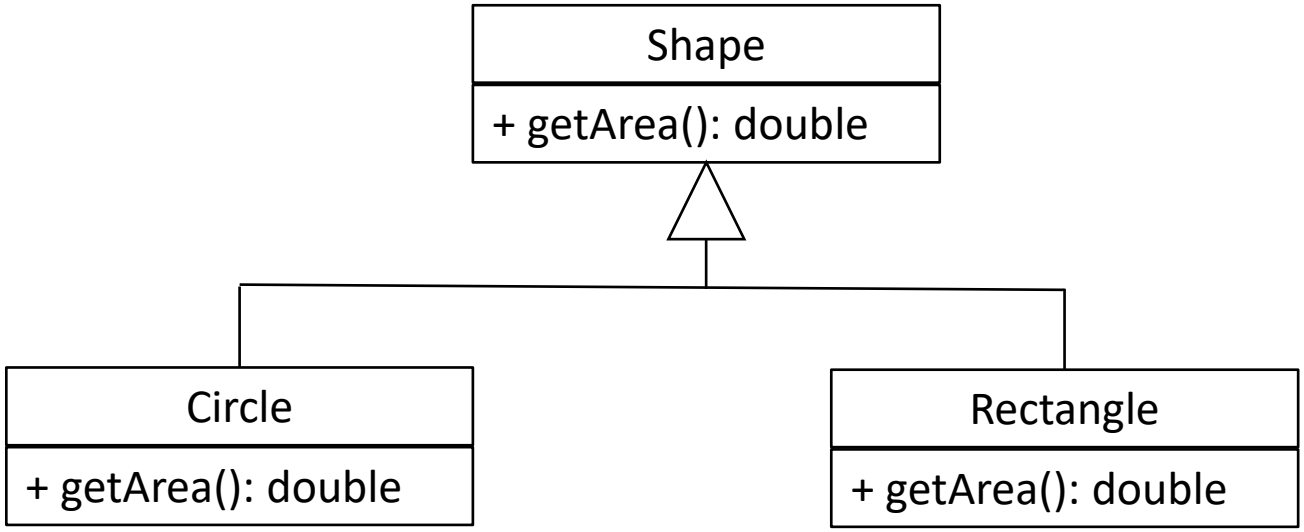
Outline

- Recap
  - Inheritance and polymorphism
- Abstract method and class

Recap: Inheritance and Polymorphism

- Problem: leveraging on polymorphism, write generic method to compute total areas of a list of geometric shapes.

The Shape Class Hierarchy



Recap: Inheritance and Polymorphism

- Problem: leveraging on polymorphism, write generic method to compute total areas of a list of geometric shapes.
- Solution

```
public double sumAreasOfShapes(ArrayList<Shape> shapeList) {
  double sum = 0.;
  for(Shape shape: shapeList) {
     sum += shape.getArea();
  }
  return sum;
}
```

The Dilemma

- Problem: leveraging on polymorphism, write generic method to compute total areas of a list of geometric shapes.
- Dilemma: but can we compute a shape's area without knowing the specification of the shape (e.g., type of shape, parameters of the shape)?

The Shape Class

- Do you like the getArea() method here?

```
public class Shape {                    …

  public double getArea() {

     throw new UnsupportedOperationException("Expect a concrete shape to compute the area");

  }

}
```

- Remarks
  - We know semantically that we can compute the area of a shape, and so we design the Shape class to have a behavior to compute its area
  - However, we don't know the algorithm to compute the area without knowing the actual shape
  - The "dummy" method in the above is not only semantically undesired, but also can easily cause runtime errors.

Abstract Method

- An abstract method has no implementation

```
public abstract double getArea() ;
```

Abstract Method: No Implementation

- Let's declare an abstract method. How about these code snippets (which one is correct or wrong and why?)

```
abstract double getArea() ;
```

```
double getArea() ;
```

```
abstract double getArea() {
}
```

```
abstract double getArea() {}
```

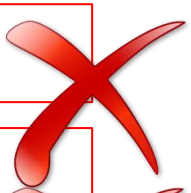
```
double getArea() {}
```

Abstract Method: No Implementation

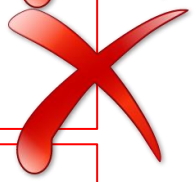
- Let's declare an abstract method. How about these code snippets?

```
abstract double getArea() ;
```

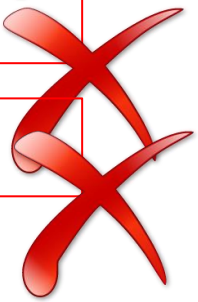
```
double getArea() ;
```



```
abstract double getArea() {
}
```



```
abstract double getArea() {}
```



```
double getArea() {}
```

Abstract Class

- In Java, any class that has an abstract method must be declared “abstract”
- Example

```
abstract class Shape {
   public abstract double area() ;
}
```

- Abstract class: a class that is declared abstract
- Abstract classes cannot be instantiated, but they can be subclassed.

Class with Abstract Method

- Abstract method: a method that is declared without an implementation

  abstract void makeNoise();

- A class that has an abstract method must be declared abstract
  - How about these two code snippets?

```
class Animal {
   abstract void makeNoise();
}
```

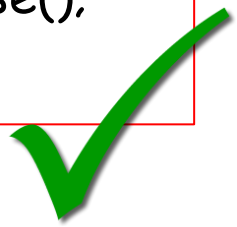
```
abstract class Animal {
   abstract void makeNoise();
}
```

Class with Abstract Method

- A class that has an abstract method must be declared abstract

```
class Animal {
   abstract void makeNoise();
}
```

```
abstract class Animal {
   abstract void makeNoise();
}
```



Abstract Class: Subclass & Instantiation

- Abstract classes ***cannot*** be instantiated, but they ***can be*** subclassed.

```
abstract class Shape {
 public abstract double area() ;
}
```

- How about these code snippets?

```
Shape s = new Shape();
```

```
class Circle extends Shape {…}
Shape s = new Circle();
```

Abstract Class: Subclass & Instantiation

- Abstract classes ***cannot*** be instantiated, but they ***can be*** subclassed.

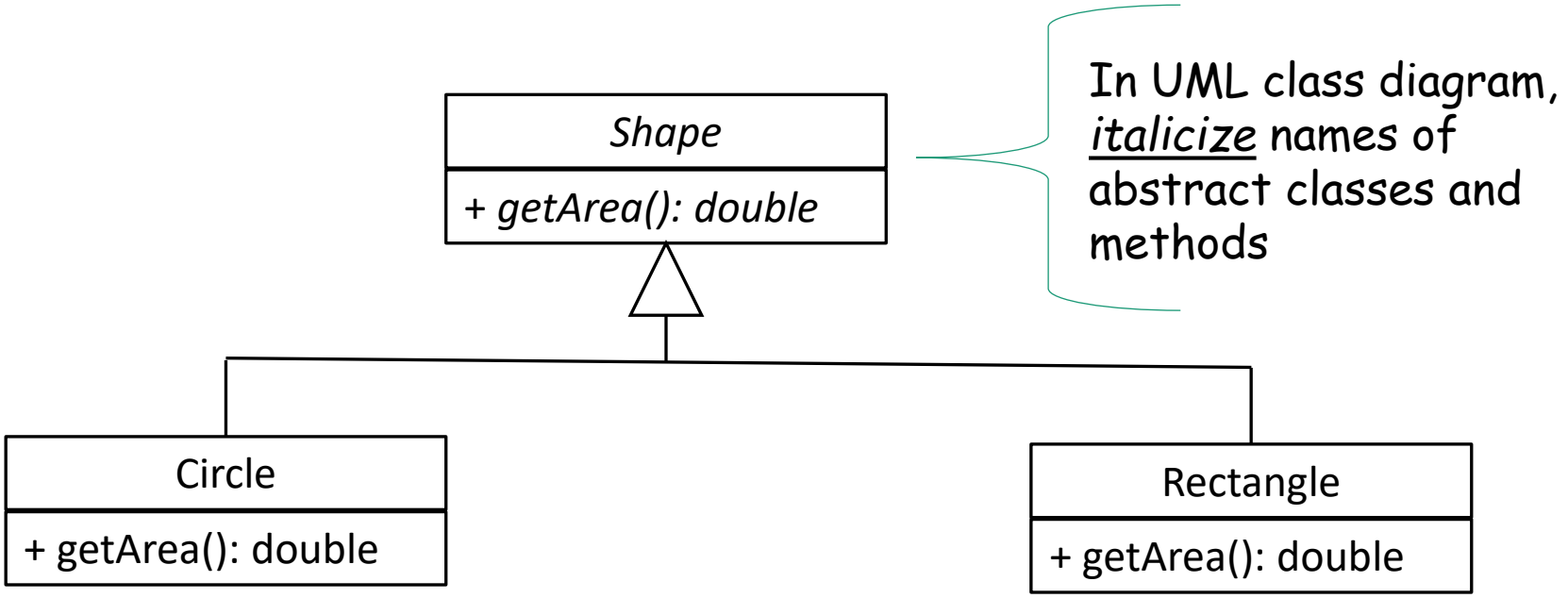
```
abstract class Shape {
 public abstract double area() ;
}
```

- How about these code snippets?

```
Shape s = new Shape();
```

```
class Circle extends Shape {…}
Shape s = new Circle();
```

The Shape Class Hierarchy: Abstract Shape



Subclass an Abstract Class

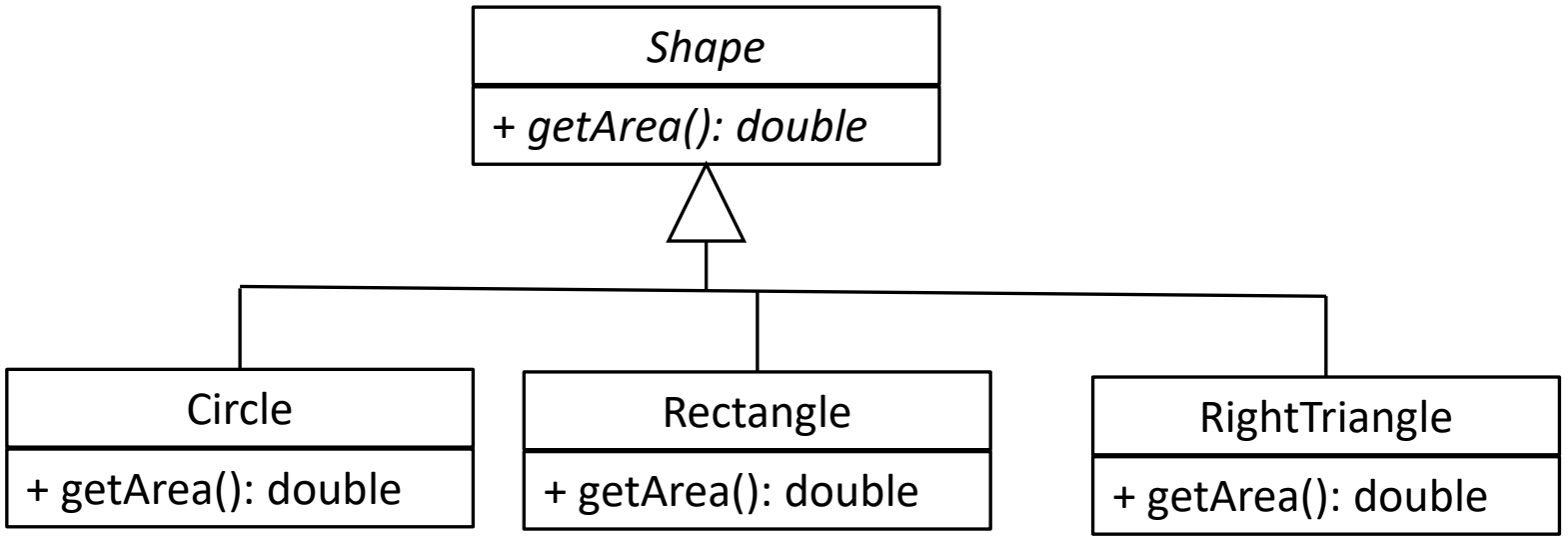
- Concrete subclass
  - A subclass may provide implementations for all of the abstract methods in its parent class.
- Abstract subclass
  - The subclass must also be declared abstract if it does not provide implementation of all of the abstract methods in its parent class.
  - The subclass may also declare abstract method itself

Concrete Subclass

- A subclass provides implementations of all of the abstract methods declared in its superclass.
- An abstract method thus can have many implementations.

The Shape Class Hierarchy: Concrete Subclasses

- Abstract class Shape's getArea method has many implementations in the abstract class's subclasses



Concrete Subclass

- Example:

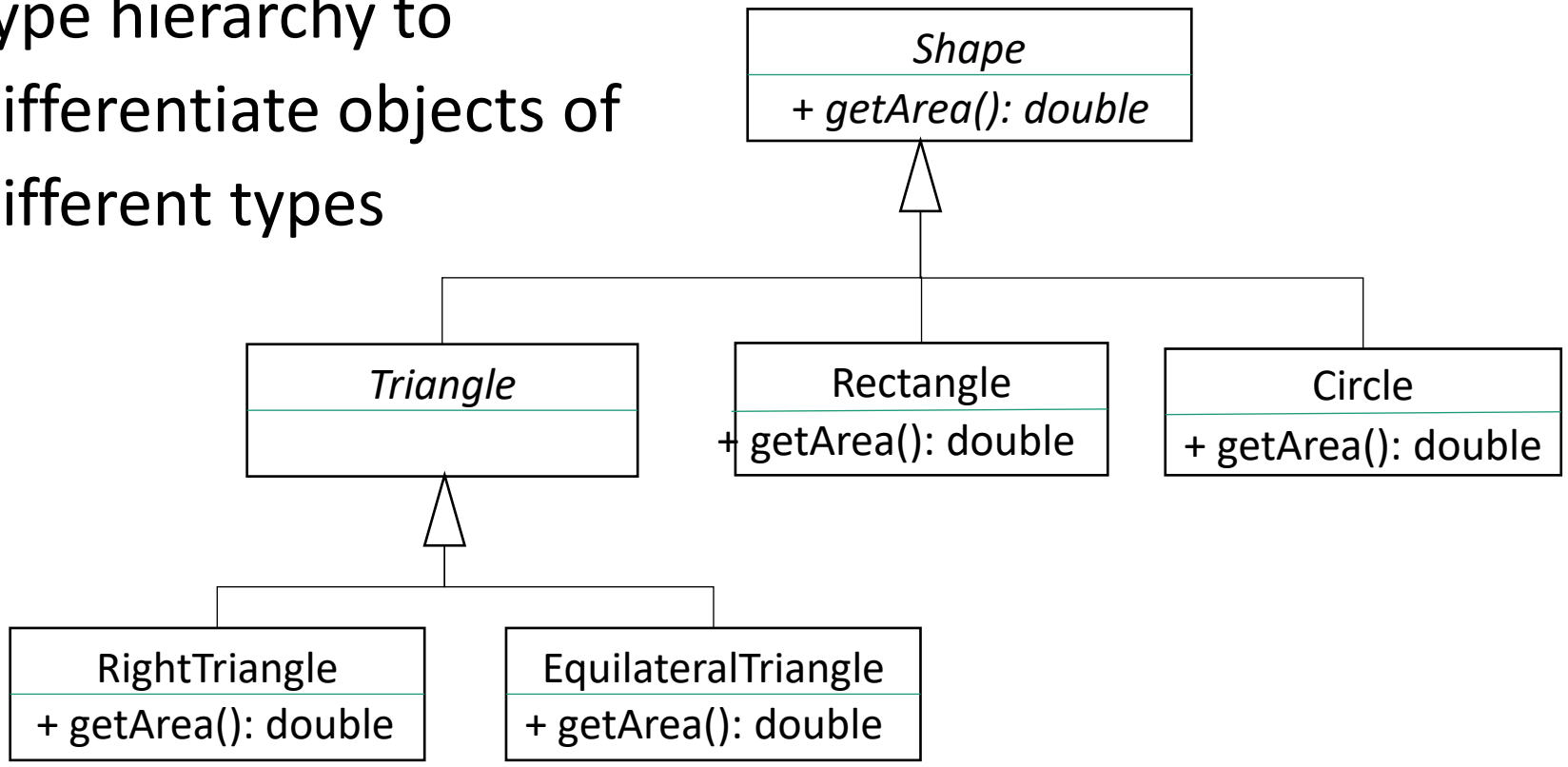
```
public class RightTriangle extends Shape {
  private double base;
  private double height;
  ……
  public double getArea() {
    return 0.5 * base * height;
  }
}
```

Abstract Subclass

- The subclass must also be declared abstract if it does not provide implementation of all of the abstract methods in its superclass.
- The subclass may also declare abstract method itself.

Abstract Subclass: Extending Type Hierarchy

- Motivation: add types to type hierarchy to differentiate objects of different types



Abstract Subclass: Extending Type Hierarchy

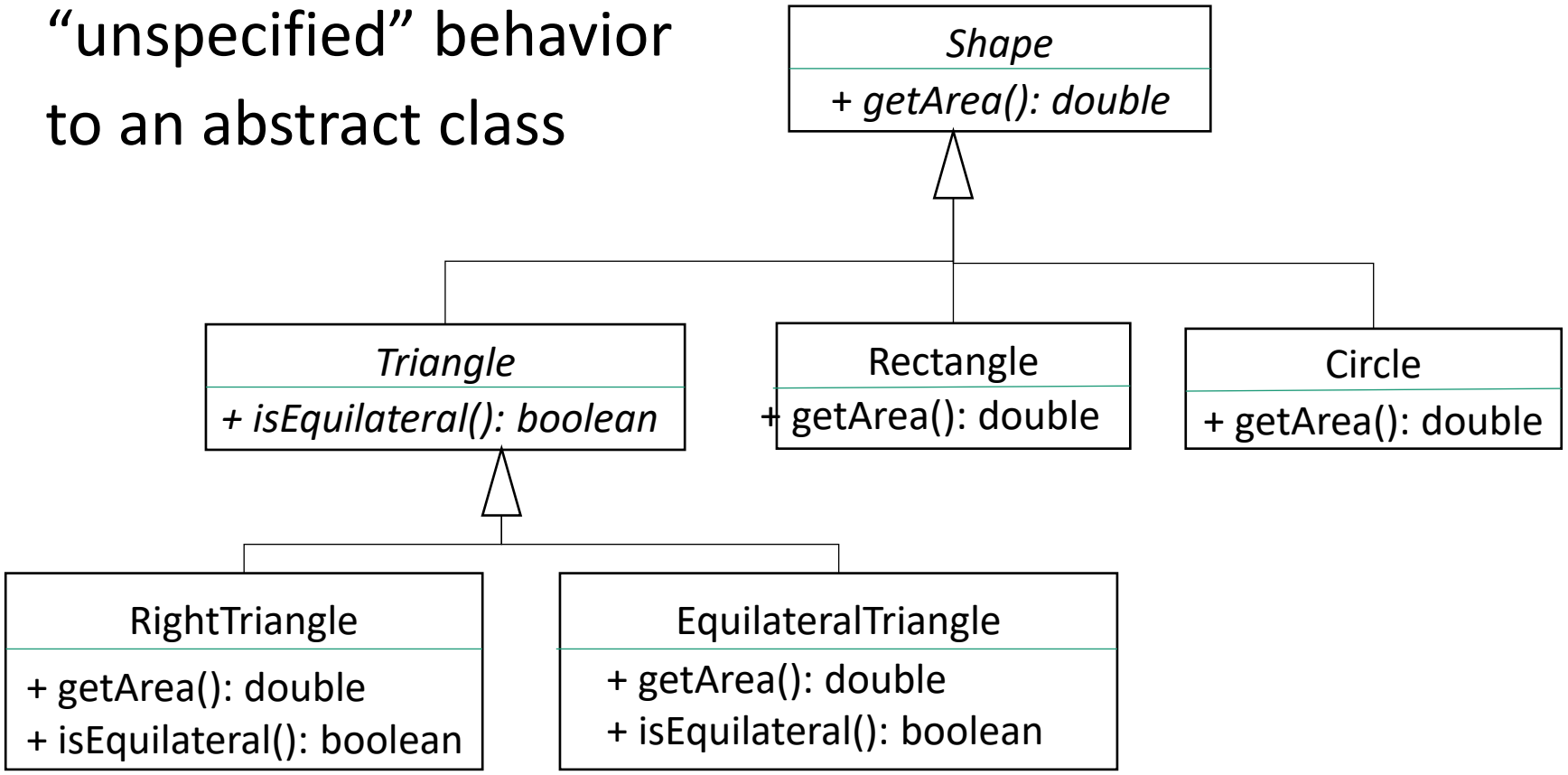
- Example:

```
public abstract class Triangle extends Shape {
    public Triangle(String name) {
        super(name);
    }
    public int getNumberOfSides() {
        return 3;
    }
}
```

- Remark:
  - An abstract class must be declared abstract.
  - An abstract class can have concrete methods.
  - An abstract class may not have any abstract methods.

Abstract Subclass: Add New Abstract Behavior

- Motivation: add new, but "unspecified" behavior to an abstract class



Abstract Subclass: Add New Behavior

- Example:

```
public abstract class Triangle extends Shape {
        public Triangle(String name) {
                super(name);
        }
        public int getNumberOfSides() {
                return 3;
        }
        public abstract boolean isEquilateral();
}
```

Questions?

- Abstract class
- Abstract method
- Extending abstract class
  - Concrete subclass
  - Abstract subclass
- Example programs

Exercise 1 of 3

- In this exercise and the exercise that follows, you are to compare the Shape class hierarchies **with** and **without** making Shape class abstract. In this exercise, you **do not** make Shape class abstract.
  - Create a **ConcreteShape** subdirectory
  - Create the following classes in the ConcreteShape directory
    - Shape, Triangle, Rectangle, Circle, RightTriangle, EquilateralTriangle, and ShapeClient classes
    - These classes are of the same class hierarchy as shown in the lecture, however, none of them are abstract class. The getArea() method is a "**dummy"** method as shown in the lecture.
    - In the ShapeClient class, generate a few shapes of different types, write method of totalArea(Shape[] shapes), and invoke it compute the sum of the areas of the shapes

Exercise 2 of 3

- In this exercise and the exercise that follows, you are to compare the Shape class hierarchies **with** and **without** making Shape class abstract. In this exercise, you **do** make Shape class abstract.
  - Create a **AbstractShape** subdirectory
  - Create the following classes in the **AbstractShape** directory
    - Shape, Triangle, Rectangle, Circle, RightTriangle, EquilateralTriangle, and ShapeClient classes
    - These classes are of the same class hierarchy as shown in the lecture. Some of these classes are abstract and the others aren't. The Shape's getArea() method is an abstract method as shown in the lecture.
    - In the ShapeClient class, generate a few shapes of different types, write method of totalArea(Shape[] shapes), and invoke it compute the sum of the areas of the shapes
    - Write a comment at the top of the Shape client class to explain the benefit of having the Shape class abstract, e.g., what kind of errors this helps prevent?

Exercise 3 of 3

- In this exercise, you are to realize the UML class diagram in next slide.
  - Create a directory for this exercise, create all the classes from scratch in the UML class diagram shown in the next slide.
    - Note that Animal and Feline are abstract classes
    - Note that makeNoise and pounce are abstract methods. To implement the methods in concrete subclasses, simply print out an appropriate message, such as, “Cat purrs”, “Panther roars”, “Dove coos”, “Whale clicks”, “Cat pounces”, “Panther pounces”, etc.
  - Write a client class, called the AnimalClient class. In the client, write three methods
    - Two generic methods, one of which takes an ArrayList of Animals and invokes each Animal’s makeNoise method, and the other of which takes an ArrayList of Felines and invokes each Feline’s pounce method
    - A main method that demonstrates the use of the two generic methods.

Exercise 3: The Animal Class Hierarchy