innovatis

Java OOP

Part 2

Encapsulation

- **Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods.**
  - The fields of a class can be made read-only or write-only.
  - Data hiding.

Example

Encapsulating the GoodDog class

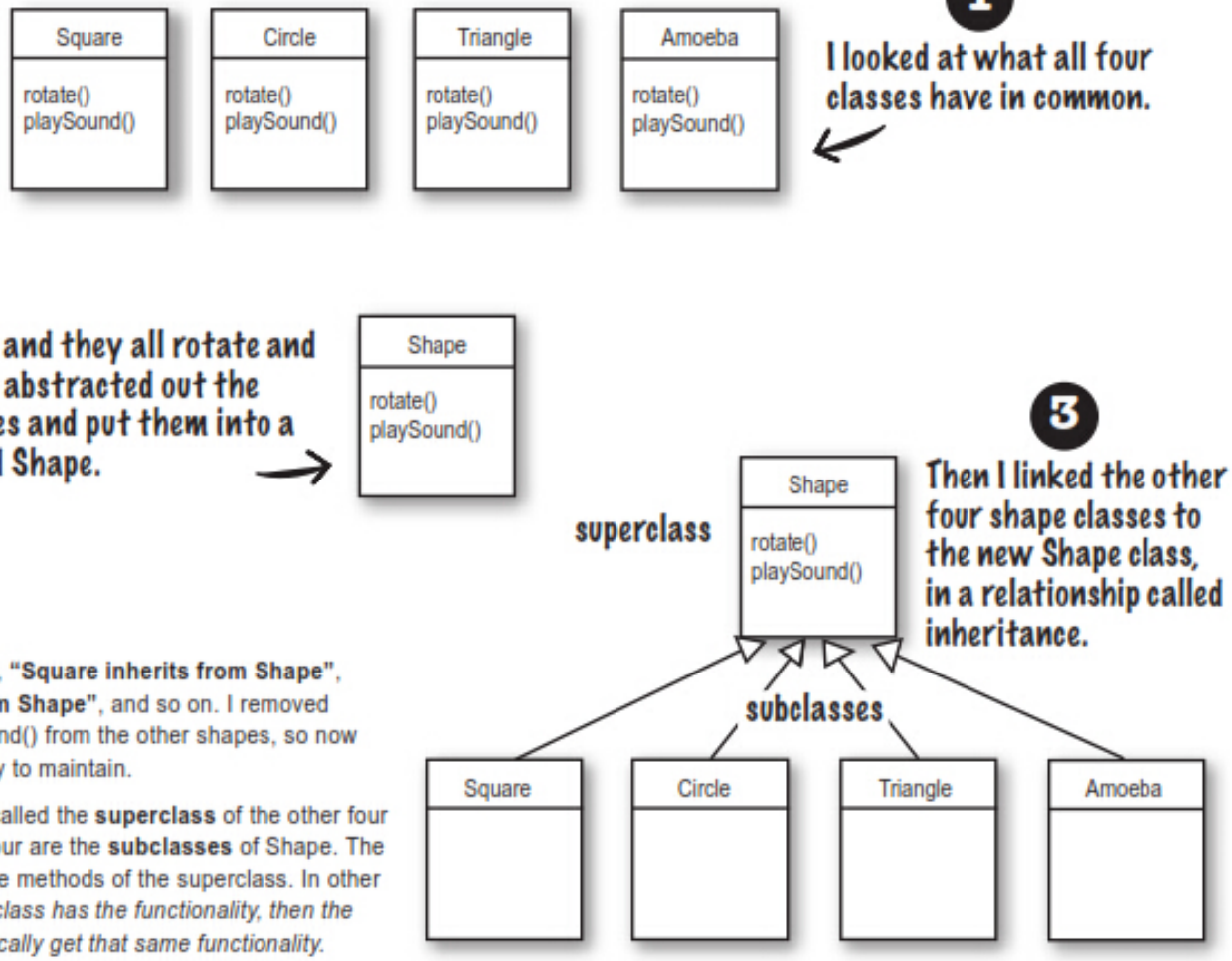
Make the instance variable private.

Make the getter and setter methods public.

Even though the methods don't really add new functionality, the cool thing is that you can change your mind later. you can come back and make a method safer, faster, better.

```
class GoodDog {

  private int size;

  public int getSize() {
    return size;
  }

  public void setSize(int s) {
    size = s;
  }

  void bark() {
    if (size > 60) {
      System.out.println("Wooof! Wooof!");
    } else if (size > 14) {
      System.out.println("Ruff!  Ruff!");
    } else {
      System.out.println("Yip! Yip!");
    }
```

Inheritance



You can read this as, **"Square inherits from Shape"**, **"Circle inherits from Shape"**, and so on. I removed rotate() and playSound() from the other shapes, so now there's only one copy to maintain.

The Shape class is called the **superclass** of the other four classes. The other four are the **subclasses** of Shape. The subclasses inherit the methods of the superclass. In other words, *if the Shape class has the functionality, then the subclasses automatically get that same functionality.*

Polymorphism

- The dictionary definition of polymorphism refers to a principle in biology in which an organism or species can have many different forms or stages.
- Subclasses of a class can define their own unique behaviors and yet share some of the same functionality of the parent class.
- Polymorphism is the capability of an action or method to do different things based on the object that it is acting upon. Overloading and overriding are two types of polymorphism.

Example: Bikes

- Note that once again, the printDescription method has been overridden. This time, information about the tire width is displayed.
- To summarize, there are three classes: Bicycle, MountainBike, and RoadBike. The two subclasses override the printDescription method and print unique information.

Example Bikes

- Bike is in gear 1 with a cadence of 20 and travelling at a speed of 10.
- Bike is in gear 5 with a cadence of 20 and travelling at a speed of 10.

  The MountainBike has a Dual suspension.
- Bike is in gear 8 with a cadence of 40 and travelling at a speed of 20.

  The RoadBike has 23 MM tires.

Overloading VS Overriding

- In a class, if two methods have the same name and a different signature, it is known as overloading in Object oriented concepts.

Overloading VS Overriding

- **When a class defines a method using the same name, return type, and arguments as a method in its superclass, the method in the class overrides the method in the superclass. When the method is invoked for an object of the class, it is the new definition of the method that is called, and not the method definition from superclass.**

- **Rules to follow :**
  - Methods may be overridden to be more public, not more private.
  - Methods may be overridden to be throws the same exception or subclass of the exception thrown by the method of superclass

Abstract class

- An *abstract class* is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.
- An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

```
abstract void moveTo(double deltaX, double deltaY);
```

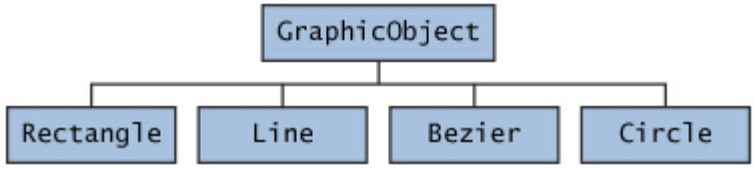
- If a class includes abstract methods, then the class itself *must* be declared abstract, as in:

```
public abstract class GraphicObject {
    // declare fields
    // declare nonabstract methods
    abstract void draw(); }
```

Abstract class

- When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. **However, if it does not, then the subclass must also be declared abstract.**

Abstract class example




```
abstract class GraphicObject {
    int x, y;
    ...
    void moveTo(int newX, int newY) {
        ...
    }
    abstract void draw();
    abstract void resize();
}
```

```
class Circle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
class Rectangle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
```

Interface

- The interface body can contain abstract methods, default methods, and static methods.
- All constant values defined in an interface are implicitly **public**, **static**, and **final**. Once again, you can omit these modifiers.

Interface example

```java
/* File name : Animal.java */
interface Animal {

   public void eat();
   public void travel();
}
```

```java
/* File name : MammalInt.java */
public class MammalInt implements Animal{

   public void eat(){
      System.out.println("Mammal eats");
   }

   public void travel(){
      System.out.println("Mammal travels");
   }

   public int noOfLegs(){
      return 0;
   }

   public static void main(String args[]){
      MammalInt m = new MammalInt();
      m.eat();
      m.travel();
   }
}
```

When to use Abstract class vs Interface

- Which should you use, abstract classes or interfaces?

Consider using abstract classes if any of these statements apply to your situation:

- You want to share code among several closely related classes.
- You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
- You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.

Consider using interfaces if any of these statements apply to your situation:

- You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and Cloneable are implemented by many unrelated classes.
- You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.
- You want to take advantage of multiple inheritance of type.

Abstract class vs Interface

- interface contains methods that must be abstract; abstract class may contain concrete methods.
- interface contains variables that must be static and final; abstract class may contain non-final and final variables.
- members in an interface are public by default, abstract class may contain non-public members.
- interface is used to "implements"; whereas abstract class is used to "extends".
- interface can be used to achieve multiple inheritance; abstract class can be used as a single inheritance.
- interface can "extends" another interface, abstract class can "extends" another class and "implements" multiple interfaces.
- interface is absolutely abstract; abstract class can be invoked if a main() exists.
- interface is more flexible than abstract class because one class can only "extends" one super class, but "implements" multiple interfaces.
- If given a choice, use interface instead of abstract class.