

# Abstract Classes

In software engineering, it's common for different groups of programmers to design "contracts" that spell out how their software interacts.

Manager:

Recorder:

Presenter:

Reflector:

## Content Learning Objectives

*After completing this activity, students should be able to:*

- Generalize multiple classes that have overlapping code.
- Explain the requirements of abstract classes and methods.
- Discuss how polymorphism can be done using interfaces.

## Process Skill Goals

*During the activity, students should make progress toward:*

- Analyze compiler error messages to reach a conclusion. (Critical Thinking)



# Model 1 Loud Toys

```
public class ToySheep {  
    private int volume;  
  
    public ToySheep() {  
        this.volume = 3;  
    }  
  
    public int getVolume() {  
        return volume;  
    }  
  
    public void setVolume(int volume) {  
        this.volume = volume;  
        makeNoise();  
    }  
  
    public void makeNoise() {  
        System.out.println("Baaa");  
    }  
}
```



```
public class ToyRobot {  
    private int chargeLevel;  
    private int volume;  
  
    public ToyRobot() {  
        this.chargeLevel = 5;  
        this.volume = 10;  
    }  
  
    public void recharge() {  
        chargeLevel = 10;  
    }  
  
    public int getVolume() {  
        return volume;  
    }  
  
    public void setVolume(int volume) {  
        this.volume = volume;  
        makeNoise();  
    }  
  
    public void makeNoise() {  
        System.out.println("Beep Beep!");  
    }  
}
```

## Questions (15 min)

Start time:

1. Identify *similarities* in the code:
  - a) What attributes do the classes have in common?
  - b) What methods do the classes have in common?
2. Summarize *differences* between the constructors and the makeNoise methods.

3. Design a new class named LoudToy that contains the code that ToySheep and ToyRobots have in common. The constructor of LoudToy should take volume as a parameter. The makeNoise method should have an empty body.

```
public class LoudToy {
```

```
}
```

4. Redesign ToySheep so that it extends LoudToy. The constructor of ToySheep should call the constructor of LoudToy. Remove the code from ToySheep that is no longer necessary.

```
public class ToySheep extends LoudToy {
```

```
}
```

5. Redesign ToyRobot so that it extends LoudToy. Remove the code from ToyRobot that is no longer necessary.

```
public class ToyRobot extends LoudToy {
```

```
}
```

6. What is the output of the following examples?

a) LoudToy toy1 = new LoudToy(1);  
toy1.makeNoise();

b) LoudToy toy2 = new ToySheep();  
toy2.makeNoise();

c) LoudToy toy3 = new ToyRobot();  
toy3.makeNoise();

7. In the previous question, did the variable's type or the object's type determine the version of makeNoise that was called?

## Model 2 Abstract Methods

The `abstract` keyword can be used to declare methods that have no body. Classes with abstract methods must also be defined as abstract.

```
public abstract class LoudToy {
    private int volume;

    public LoudToy(int volume) {
        this.volume = volume;
    }

    public int getVolume() {
        return volume;
    }

    public void setVolume(int volume) {
        this.volume = volume;
        makeNoise();
    }

    public abstract void makeNoise();
}
```

### Questions (15 min)

**Start time:**

8. Summarize the differences between Model 2 and your answer to #3.
9. Open *LoudToy.java* (from Model 2) in your IDE. Remove the word `abstract` from the class definition. What are the two compiler errors?
10. Replace the word `abstract` in the class definition, and then remove the word `abstract` from the method definition. What is the compiler error now?

11. Remove the definition of `makeNoise` altogether, and notice the compiler error. Why is it necessary to declare this method in `LoudToy`?

12. Undo all changes in `LoudToy.java`, and add the following main method. What is the compiler error message? Why do you think Java doesn't allow you to construct a `LoudToy`?

```
public static void main(String[] args) {  
    LoudToy toy1 = new LoudToy(1);  
    toy1.makeNoise();  
}
```

13. Open `ToySheep.java` and rename `makeNoise` to `makeNoise2`. What is the compiler error?

14. Rename the method back to `makeNoise`, but change `void` to `int`. What is the error now?

15. Explain how an abstract method is like a contract.

## Model 3 Java Interfaces

An interface is similar to an abstract class, except that all methods are automatically **public** and **abstract**. Likewise, all fields are automatically **public**, **static**, and **final**. These keywords are omitted in the interface definition.

```
public interface Rechargeable {  
    int MAX_CHARGE = 10;  
  
    int getCharge();  
  
    void recharge();  
}
```

Classes do not extend interfaces; they implement them:

```
public class CellPhone implements Rechargeable {  
    private int chargeLevel;  
    private int volume;  
  
    public CellPhone(int chargeLevel, int volume) {  
        this.chargeLevel = chargeLevel;  
        this.volume = volume;  
    }  
  
    public int getCharge() {  
        return chargeLevel;  
    }  
  
    public void recharge() {  
        chargeLevel = MAX_CHARGE;  
    }  
  
    public int getVolume() {  
        return volume;  
    }  
  
    public void setVolume(int volume) {  
        this.volume = volume;  
    }  
  
    public void makeCall() {  
        System.out.println("Ring... Hello?");  
    }  
}
```

## Questions (15 min)

Start time:

16. What two methods are required by Rechargeable?

17. Modify your *ToyRobot.java* to implement the Rechargeable interface. What changes did you need to make?

18. Consider the following rechargeAll method. What type of objects are stored in the list?

```
public static void rechargeAll(ArrayList<Rechargeable> list) {  
    for (Rechargeable item : list) {  
        item.recharge();  
    }  
}
```

19. Consider the following main method. Explain the significance of storing ToyRobot and CellPhone objects in the same ArrayList when calling rechargeAll.

```
public static void main(String[] args) {  
    ArrayList<Rechargeable> items = new ArrayList<>();  
    items.add(new ToyRobot());  
    items.add(new CellPhone(4, 5));  
    rechargeAll(items);  
}
```

20. Explain how an interface is like a contract.