Abstract Methods, Abstract Classes, and Interfaces

Weiss, sections 4.2 through 4.5 and section 4.7 (pages 125–141 and 150–157)
Abstract Class

- Consider graphical objects like rectangles, lines, circles, and curves.
  - These objects have behaviors /methods (e.g., moveTo, rotate, resize, draw) and states (e.g., position, fill color)
  - Some behaviors are different for different objects: resize or draw
  - Each object has a different implementation for those behaviors.

- An super class declares resize and draw without implementation.
- Subclasses of the super class implement resize and draw.
Abstract Class

- **Abstract class** is not instantiated.
- Abstract class can be subclassed.
- Subclasses must implement all **abstract methods**. Otherwise, it is declared as an abstract class.

```java
abstract class GraphicalObject {
    int x, y;

    void moveTo(int newX, int newY) {
    }

    abstract void draw();
    abstract void resize();
}
```

```java
class Circle extends GraphicalObject {
    void draw() {
    }

    void resize() {
    }
}
```

```java
class Rectangle extends GraphicalObject {
    void draw() {
    }

    void resize() {
    }
}
```
Abstract Class

• Abstract class can have constructors. The constructors are called by subclasses using `super`.
  
➢ The constructors should be defined as `protected`.

```java
abstract class GraphicalObject {
    int x, y;

    protected GraphicalObject(int x, int y) {
        this.x = x;
        this.y = y;
    }

    void moveTo(int newX, int newY) {
    }

    abstract void draw();
    abstract void resize();
}

class Rectangle extends GraphicalObject {
    public Rectangle(int x, int y) {
        super(x, y);
    }

    void draw() {
    }

    void resize() {
    }
}
```
Interface

• Interface is a **contract** between disparate groups of programmers that specifies how their programs interact together.

• Consider self-driving cars
  • Car manufacturers write programs (e.g., methods) that move cars: turn left/right, stop, start, accelerate.
  • Electronic manufacturers write programs that use GPS position and traffic data to move cars (invoke methods).
  • Car manufacturers set up an industry-standard **interface** that specifies what methods are called to move the car.

• Neither manufacturers need to know how the other manufacturer’s program is written.
Interface

- Interface is a reference type. It contains *abstract* methods and constants.
- Interfaces are not instantiated. They are implemented by other classes or extended by other interfaces.
- All interface methods must be *public*.
- A class can implement more than one interfaces. This is called *multiple inheritance* in Java language.

```java
public interface OperateCar {
    public static final int OPERATED_CARS = 1000;
    enum Direction {LEFT, RIGHT};
    public void turn(Direction direction, double radius, double startSpeed, double endSpeed);
    public void signalTurn(Direction direction, boolean signalOn);
}

public class OperateBMW760i implements OperateCar {
    public void turn(Direction direction, double radius, double startSpeed, double endSpeed) {};
    public void signalTurn(Direction direction, boolean signalOn) {};
}
```
Consider an interface that defines how to compare the size of objects.

- The size of geometric objects are compared using area.
- When any class implements Relatable, the size of any two objects of that class are comparable.

```java
public interface Relatable {
    public int isLargerThan(Relatable other);
}
```
public class RectanglePlus implements Relatable {
    int width;
    int height;
    Point point;

    public RectanglePlus(Point p, int w, int h) {
        this.point = p;
        this.width = w;
        this.height = h;
    }

    public int getArea() {
        return width * height;
    }

    public int isLargerThan(Relatable other) {
        RectanglePlus otherRect = (RectanglePlus)other;
        if (this.getArea() < otherRect.getArea()) {
            return -1;
        } else if (this.getArea() > otherRect.getArea()) {
            return 1;
        } else {
            return 0;
        }
    }
}
Interface

• If you define a reference variable of an interface type, any object you assign to it must be an instance of a class that implements the interface.

➢ *Type compatibility* applies between an interface and a class that implements the interface.

```java
Point p1 = new Point(0, 0);
RectanglePlus rect1 = new RectanglePlus(p1, 20, 3);

Point p2 = new Point(3, 3);
RectanglePlus rect2 = new RectanglePlus(p2, 20, 3);

rect1.isLargerThan(rect2);
```