Fundamentals of Java

Inheritance

What is inheritance?

So far, all of our classes have been written from scratch.

It is possible to write a *subclass* that is an extension of an existing class (the *superclass*).

- A subclass automatically "inherits" all fields and methods of the superclass.
- A subclass may then define more fields and methods in addition to the inherited fields and methods.

Example superclass

```
public class Rectangle
{
    private int width;
    private int height;
}
```

Example subclass

```
public class Rectangle {
    private int width;
    private int height;
}
public class PositionedRectangle extends Rectangle {
    private int x;
    private int y;
}
```

Example subclass

```
public class PositionedRectangle extends Rectangle
{
    private int x;
    private int y;
}
```

This is similar to writing:

```
public class PositionedRectangle
{
    private int width;
    private int height;
    private int x;
    private int y;
}
```

Example superclass (continued)

```
public class Rectangle {
    ...

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

Example subclass (continued)

```
public class PositionedRectangle extends Rectangle {
    ...
    public void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
}
```

Positioned rectangle now has two methods:

- getArea -- which is inherited
- move

Example superclass (continued)

```
public class Rectangle {
    ...
    public void show() {
        System.out.println("Width: " + width);
        System.out.println("Height: " + height);
    }
}
```

If a new method is added to Rectangle, it is also automatically inherited into PositionedRectangle.

Method overloading

```
public class PositionedRectangle extends Rectangle {
    ...
    public void show() {
        super.show();
        System.out.println("X: " + x);
        System.out.println("Y: " + y);
    }
}
```

The subclass redefines the show() method.
This is called *"method overloading"*.
super.show() calls the superclass's version of the method.

Constructors are not inherited

```
public class Rectangle
{
    private int width;
    private int height;
    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }
}
```

Constructors are not inherited

```
public class PositionedRectangle extends Rectangle {
    private int x;
    private int y;
    public PositionedRectangle(int x, int y,
                                int width, int height) {
        super(width, height);
        this.x = x;
        this.y = y;
    }
}
```

- The subclass *must* define its own constructor.
- The subclass' constructor *must* invoke the superclass' constructor via super();

Private fields are inaccessible in subclasses!

```
public class PositionedRectangle extends Rectangle
{
    private int x;
    private int y;
    public PositionedRectangle(int x, int y,
                                                          int width, int height) {
        this.width = width;    // access denied!
        this.height = height; // access denied!
        this.x = x;
        this.y = y;
    }
}
```

The subclass cannot directly access its own width and height. This is the responsibility of the superclass!

Access modifiers

- `private int x;`
  Accessible only in the same class.
- `public int x;`
  Accessible by all classes.
- `protected int x;`
  Accessible by all classes in the same package, and by subclasses in any package.
- `int x;`
  Accessible by all classes in the same package.

Protected access by subclasses

```
public class Rectangle { protected int width, height; ...
}
public class PositionedRectangle extends Rectangle {
    protected int x, y;
    ...
    public void paint(Graphics g) {
        g.drawLine(x, y, x + width, y);
        d.drawLine(x + width, y, x + width, y + height);
        d.drawLine(x + width, y + height, x, y + height);
        g.drawLine(x, y + height, x, y);
    }
}
```

The paint() method requires access to the width and height fields of the superclass so declare them "**protected**".