Building Java Programs

Chapter 8

Lecture 8-1: Classes and Objects

reading: 8.1-8.2

self-checks: Ch. 8 #1-9

exercises: Ch. 8 #1-4

Problem

- Declaring same group of related variables several times in a program

```
int x1 = 3;
int y1 = 5;
int x2 = 12;
int y2 = 4;
```

  - Annoying and redundant
  - Unclear and hard to keep track of variables

Solution: Objects

- Group together related variables into an **object**
  - Like creating your own data structure out of Java building blocks

```
public class <object name> {
    <field(s)>;
}
```

- Syntax to use this data structure:

```
<object> <variable> = new <object>();
```

Solution: Objects

- Group together related variables into an **object**
  - Like creating your own data structure out of Java building blocks

```
public class Point {
    int x;
    int y;
}
```

- Syntax to use this data structure:

```
Point p1 = new Point();
```

Two Uses for Java Classes

- **class**: A program entity that represents either:
  1. A program / module, or
  2. **A template for a new type of objects.**

  - The `DrawingPanel` class is a template for creating `DrawingPanel` objects.

- **object**: An entity that combines state and behavior

Java class: Program

- An **executable program** with a **main method**
  - Can be run; statements execute procedurally
  - What we've been writing all quarter

```
public class BMI2 {
    public static void main(String[] args) {
        giveIntro();
        Scanner console = new Scanner(System.in);
        double bmi1 = getBMI(console);
        double bmi2 = getBMI(console);
        reportResults(bmi1, bmi2);
    }
    ...
}
```

Java class: Object Definition

- A **blueprint** for a new data type
  - Not executable, not a complete program
- Created objects are an **instance** of the class

- Blueprint:

```
public class Point {
    int x;
    int y;
}
```

- Instance:

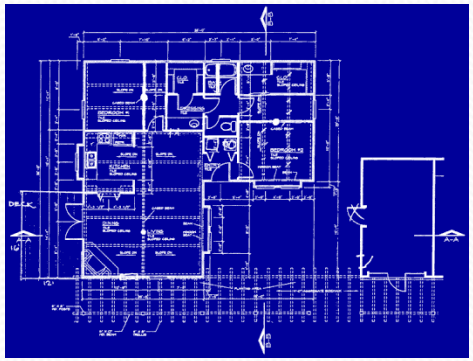
```
Point p1 = new Point();
```

Blueprint analogy

iPod blueprint

state:
- current song
- volume
- battery life

behavior:
- power on/off
- change station/song
- change volume
- choose random song



creates

iPod #1

state:
- song = "Octopus's Garden"
- volume = 17
- battery life = 2.5 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

iPod #2

state:
- song = "Lovely Rita"
- volume = 9
- battery life = 3.41 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

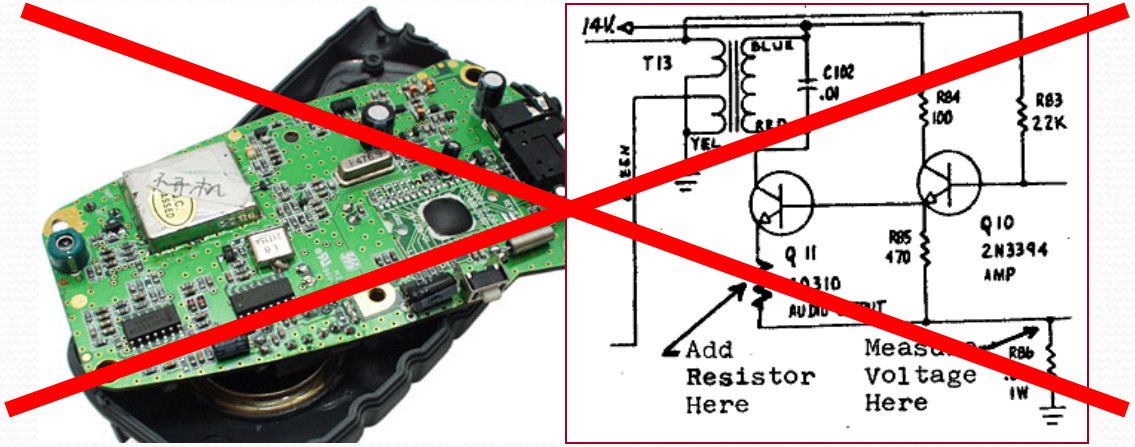
iPod #3

state:
- song = "For No One"
- volume = 24
- battery life = 1.8 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

Abstraction

- **abstraction**: A distancing between ideas and details.
  - We can use objects without knowing how they work.
- abstraction in an iPod:
  - You understand its external behavior (buttons, screen).
  - You don't understand its inner details, and you don't need to.



Client and Object Classes

- **client program**: A program that uses objects.
  - Example: HW6 `Names` is a client of `DrawingPanel` and `Graphics`.

- **object**: An entity that combines state and behavior
  - *state*: data fields
  - *behavior*: methods

The Object Concept

- **procedural programming:** Programs that perform their behavior as a series of steps to be carried out

- **object-oriented programming (OOP)**: Programs that perform their behavior as interactions between objects
  - Takes practice to understand the object concept

Fields

- **field**: A variable inside an object that is part of its state.
  - Each object has *its own copy* of each field.

- Clients can access/modify an object's fields
  - access: **<variable>.<field>**
  - modify: **<variable>.<field>** = **<value>**;

- Example:

```
Point p1 = new Point();
Point p2 = new Point();
System.out.println("the x-coord is " + p1.x);    // access
p2.y = 13;                                       // modify
```

Behavior

- Objects can tie related data and *behavior* together
- **instance method:** A method inside an object that operates on that object

```
public <type> <name> (<parameter(s)>) {
    <statement(s)>;
}
```

- Syntax to use method:
  <variable>.<method>(<parameter(s)>);
- Example:

```
p1.translate(11, 6);
```

Implicit Parameter

- Each instance method call happens on a particular object.
  - Example: `p1.translate(11, 6);`
- The code for an instance method has an implied knowledge of what object it is operating on.
- **implicit parameter**: The object on which an instance method is called.
  - Can be referred to inside the object using `this` keyword

Accessors

- **accessor**: An instance method that provides information about the state of an object.

- Example:

```
public double distanceFromOrigin() {
    return Math.sqrt(x * x + y * y);
}
```

- This gives clients "read-only" access to the object's fields.

Mutators

- **mutator**: An instance method that modifies the object's internal state.

- Example:

```
public void translate(int dx, int dy) {
      x += dx;
      y += dy;
}
```

- This gives clients both read and write access to code.