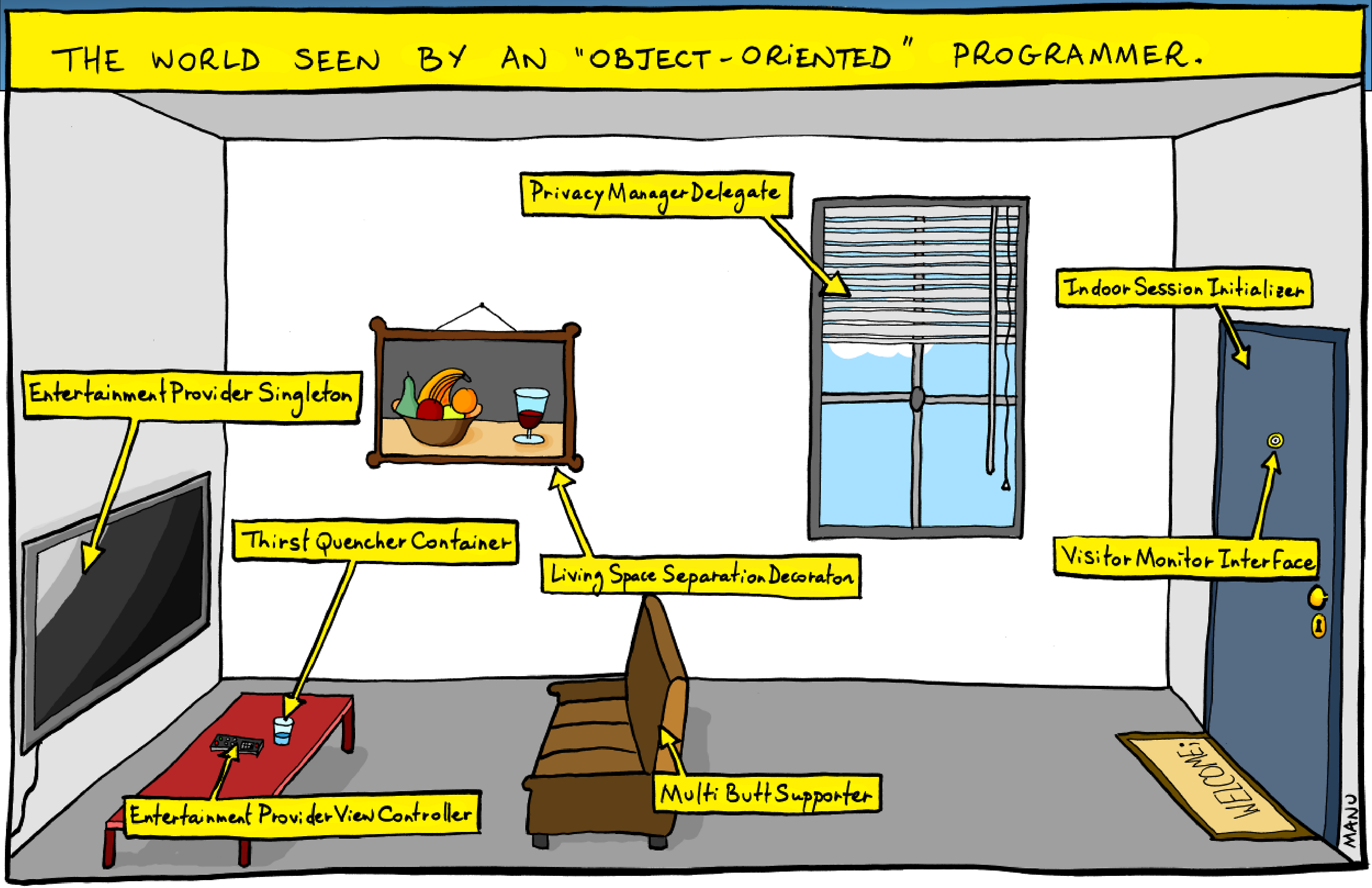
Building Java Programs
Chapter 8

Classes

CS& 141 Computer Science I Java

Green River College


THE WORLD SEEN BY AN "OBJECT-ORIENTED" PROGRAMMER.
Privacy Manager Delegate
Indoor Session Initializer
Entertainment Provider Singleton
Thirst Quencher Container
Living Space Separation Decorator
Visitor Monitor Interface
Multi Butt Supporter
Entertainment Provider View Controller
WELCOME!
MANU

So far, what we have done..

- **procedural programming:** Programs that perform their behavior as a series of steps to be carried out
  - Focuses on functions/tasks
  - As programs got bigger and more complex, difficult to manage

The Object Concept

- **procedural programming:** Programs that perform their behavior as a series of steps to be carried out
  - Focuses on functions/tasks
  - As programs got bigger and more complex, difficult to manage

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

Object

- Programming entity that contains state (data) and behavior (methods)

States: make, model, color

behaviors: start(), stop(), move()

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 `Random` class is a template for creating `Random` objects.

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

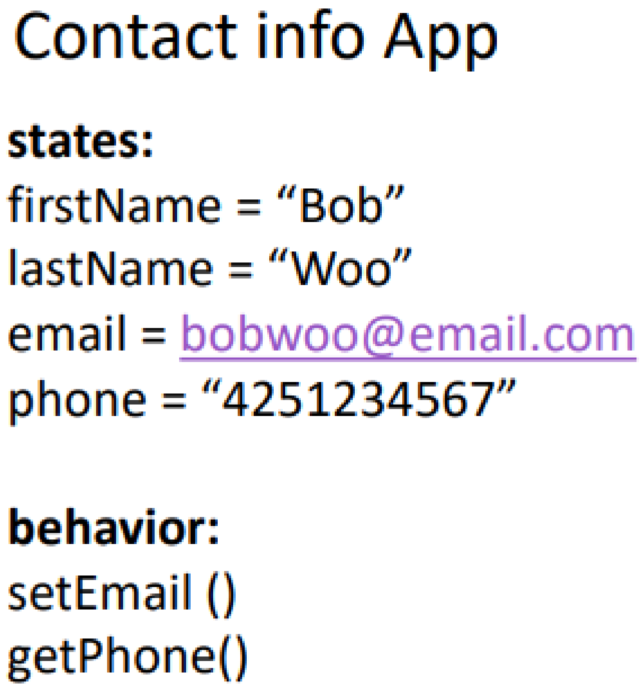
Java class: Program

- 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);
    }
    ...
}
```

Object

- Programming entity that contains state (data) and behavior (methods)



Contact info App

states:

firstName = “Bob”

lastName = “Woo”

email = bobwoo@email.com

phone = “4251234567”

behavior:

setEmail ()

getPhone()

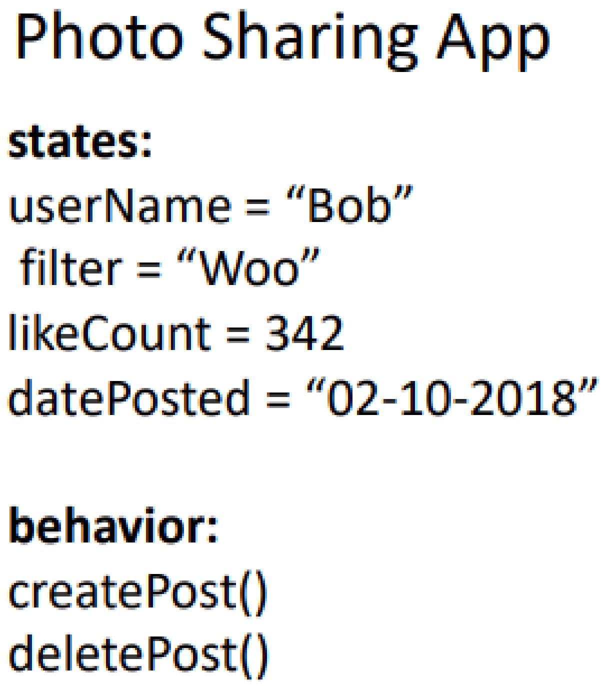


Photo Sharing App

states:

userName = “Bob”

filter = “Woo”

likeCount = 342

datePosted = “02-10-2018”

behavior:

createPost()

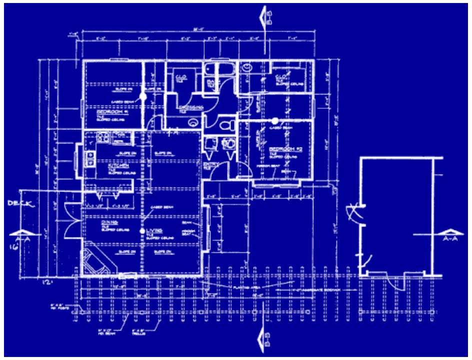
deletePost()

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

behavio
power on/c
change station/sor
change volum
choose random sor

iPod #2

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

behavior
power on/of
change station/song
change volume
choose random song

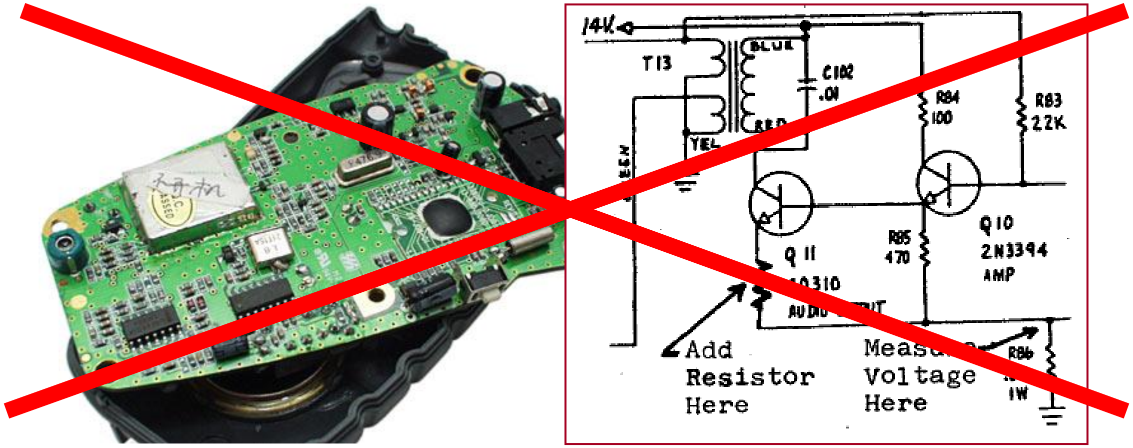
iPod #3

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

behavior
power on/of
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.



Problem

- Given a file of points and their x and y coordinates,

A	50	20
B	90	60
C	10	72
D	74	98
E	5	136
F	150	91

Write a program to find the distance from each point to the origin (0,0)

Declaring same group of related variables several times in a program

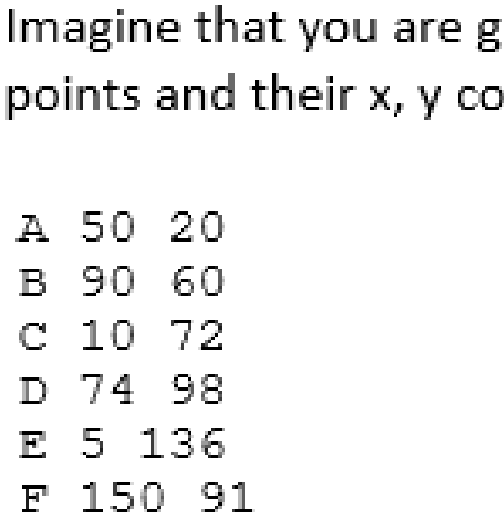
```
int x1 = 50;
int y1 = 20;
int x2 = 90;
int y2 = 60;
```

Annoying and redundant

Unclear and hard to keep track of variables

Observations

- The data in this problem is a set of points.
- It would be better stored as `Point` objects.
  - A `Point` would store a point's x/y data along with the name.
  - Point action: Each `Point` would know how to find the distance between itself and the origin
  - Point action: We could compare distances between `Point`s
  - The overall program would be shorter and cleaner.



Imagine that you are g
points and their x, y co

A	50	20
B	90	60
C	10	72
D	74	98
E	5	136
F	150	91

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();
```

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:

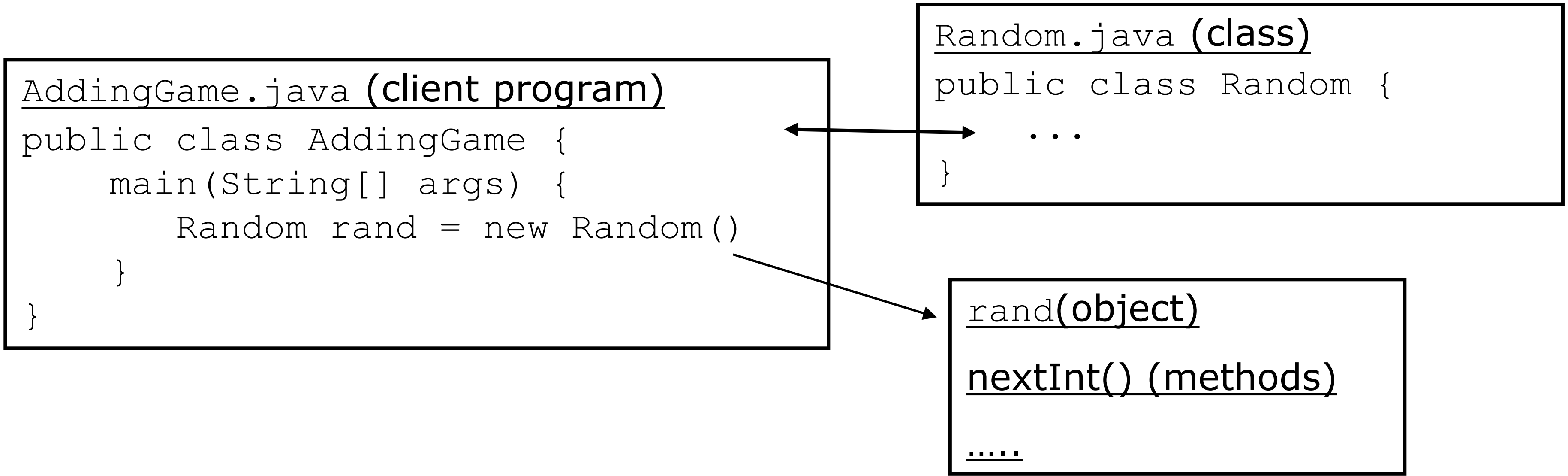
```
public class Point {
   int x;
   int y;
}
```

- Instance:

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

Building Point Class

- **Define a type of objects named Point**
  - Each `Point` object will contain x/y data called fields.
  - Each `Point` object will contain behavior called methods.
- **client program**: A program that uses objects.
  - Example: `AddingGame` is a client of `Random`.
  - Client program will use the `Point` objects



AddingGame.java (client program)

```
public class AddingGame {
    main(String[] args) {
        Random rand = new Random()
    }
}
```

Random.java (class)

```
public class Random {
      ...
}
```

rand(object)

nextInt() (methods)

…..

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);
```

Point objects (desired)

```
Point p1 = new Point(5, -2);
Point p2 = new Point();              // origin, (0, 0)
```

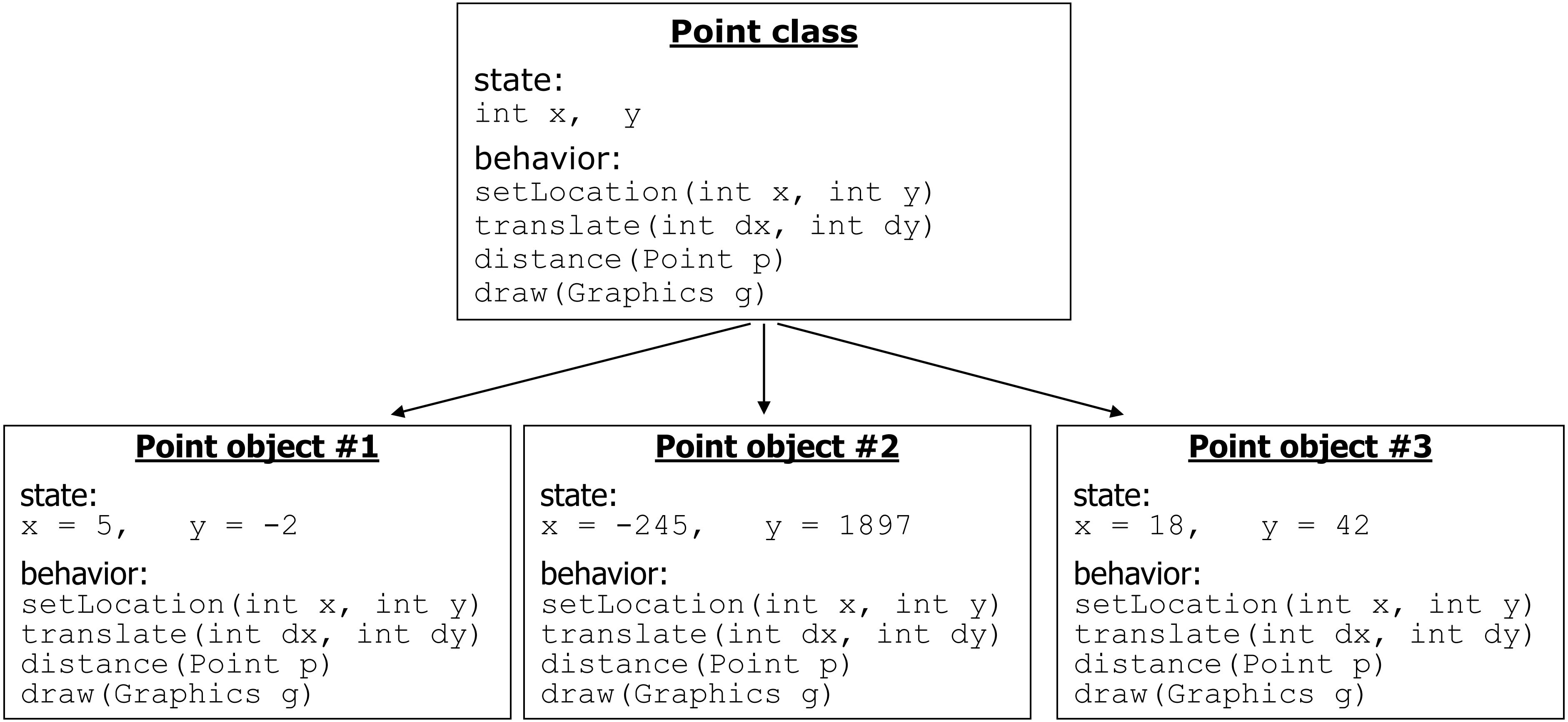
- Data in each `Point` object:

Field name	Description
`x`	the point's x-coordinate
`y`	the point's y-coordinate

- Methods in each `Point` object:

Method name	Description
`setLocation(`**x**`, `**y**`)`	sets the point's x and y to the given values
`translate(`**dx**`, `**dy**`)`	adjusts the point's x and y by the given amounts
`distance(`**p**`)`	how far away the point is from point *p*
`draw(`**g**`)`	displays the point on a drawing panel

Point class as blueprint



Point class

state:

```
int x,  y
```

behavior:

```
setLocation(int x, int y)
translate(int dx, int dy)
distance(Point p)
draw(Graphics g)
```

Point object #1

state:

```
x = 5,    y = -2
```

behavior:

```
setLocation(int x, int y)
translate(int dx, int dy)
distance(Point p)
draw(Graphics g)
```

Point object #2

state:

```
x = -245,    y = 1897
```

behavior:

```
setLocation(int x, int y)
translate(int dx, int dy)
distance(Point p)
draw(Graphics g)
```

Point object #3

state:

```
x = 18,    y = 42
```

behavior:

```
setLocation(int x, int y)
translate(int dx, int dy)
distance(Point p)
draw(Graphics g)
```

- – The class (blueprint) will describe how to create objects.
- – Each object will contain its own data and methods.

Object state: Fields

Point class, version 1

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

- Save this code into a file named `Point.java`.

- The above code creates a new type named `Point`.
  - Each `Point` object contains two pieces of data:
    - an `int` named `x`, and
    - an `int` named `y`.
  - `Point` objects do not contain any behavior (yet).

Fields

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

- Declaration syntax:

  type name;

  - Example:

```
public class Student {
    String name;    // each Student object has a
    double gpa;     // name and gpa field
}
```

Accessing fields

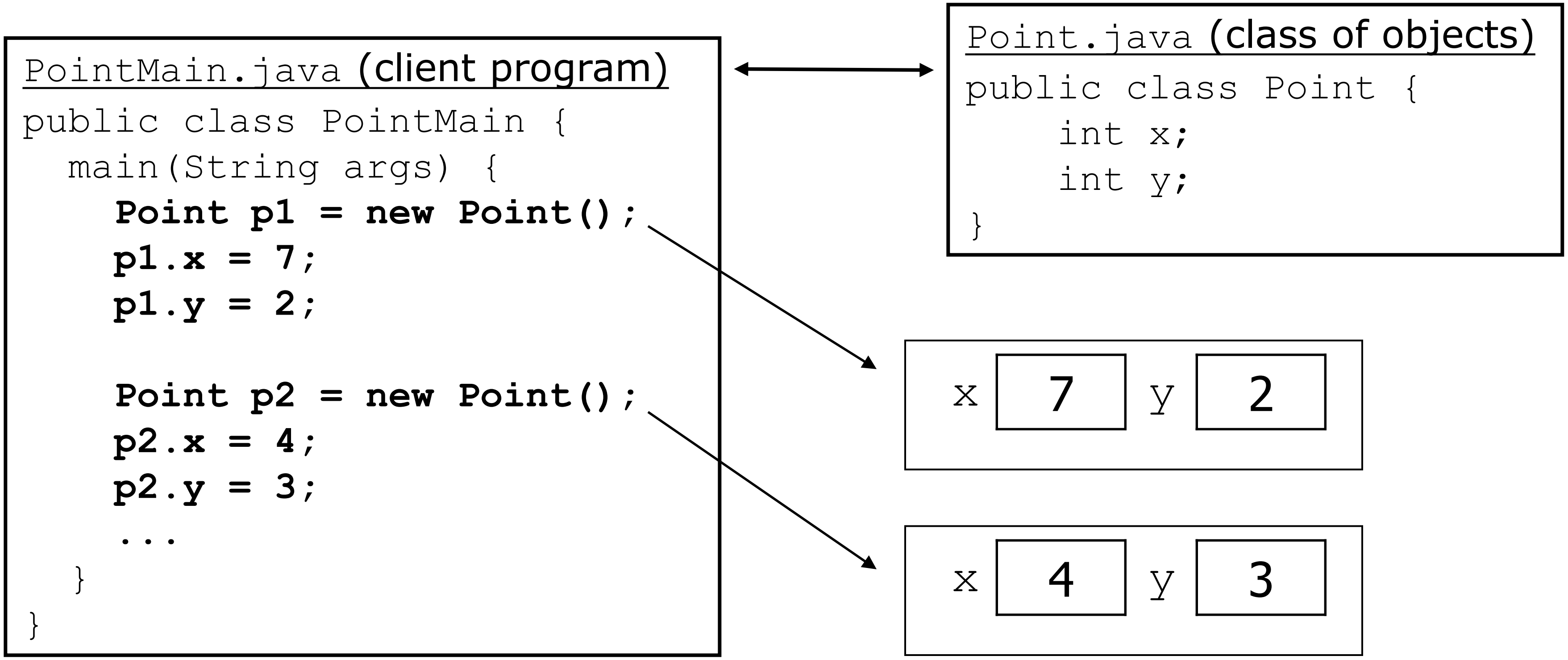
- Other classes can access/modify an object's fields.
  - access: **variable . field**
  - modify: **variable . field** = **value** ;

- Example:

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

A class and its client

- `Point.java` is not, by itself, a runnable program.
  - A class can be used by **client** programs.



`PointMain.java` (client program)

```
public class PointMain {
  main(String args) {
    Point p1 = new Point();
    p1.x = 7;
    p1.y = 2;

    Point p2 = new Point();
    p2.x = 4;
    p2.y = 3;
    ...
  }
}
```

`Point.java` (class of objects)

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

Write a program

- Implement the point class with x and y data (int)
- Write a client program to access your point class and assign data to point object you created

```
public class pointMain {
    public static void main (String [] args) {
      //create two point objects (0,2) (4,0)


      // move p2 x coordinate by 2 and y coordinate
      // by 1 and print p2
    }
}
```

- Submit on Canvas when you are done (OOP day 1 Exit Quiz)