Building Java Programs

Chapter 8

Lecture 8-1: Classes and Objects

reading: 8.1 - 8.3

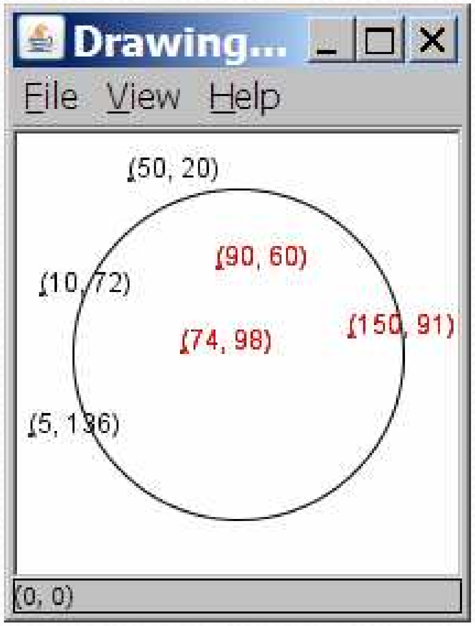
self-checks: #1-9

exercises: #1-4

A programming problem

- Given a file of cities' (x, y) coordinates, which begins with the number of cities:

```
6
50 20
90 60
10 72
74 98
5 136
150 91
```



- Write a program to draw the cities on a `DrawingPanel`, then drop a "bomb" that turns all cities red that are within a given radius:

```
Blast site x/y? 100 100
Blast radius? 75
```

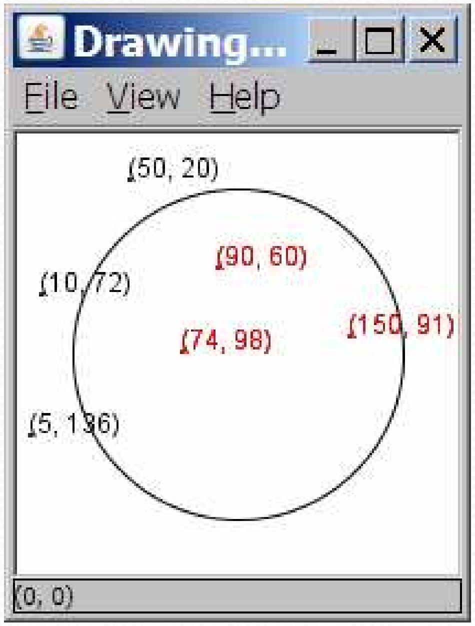
A bad solution

```
Scanner input = new Scanner(new File("cities.txt"));
int cityCount = input.nextInt();
int[] xCoords = new int[cityCount];
int[] yCoords = new int[cityCount];

for (int i = 0; i < cityCount; i++) {
    xCoords[i] = input.nextInt();   // read each city
    yCoords[i] = input.nextInt();
}
...
```

- **parallel arrays**: 2+ arrays with related data at same indexes.
  - Considered poor style.

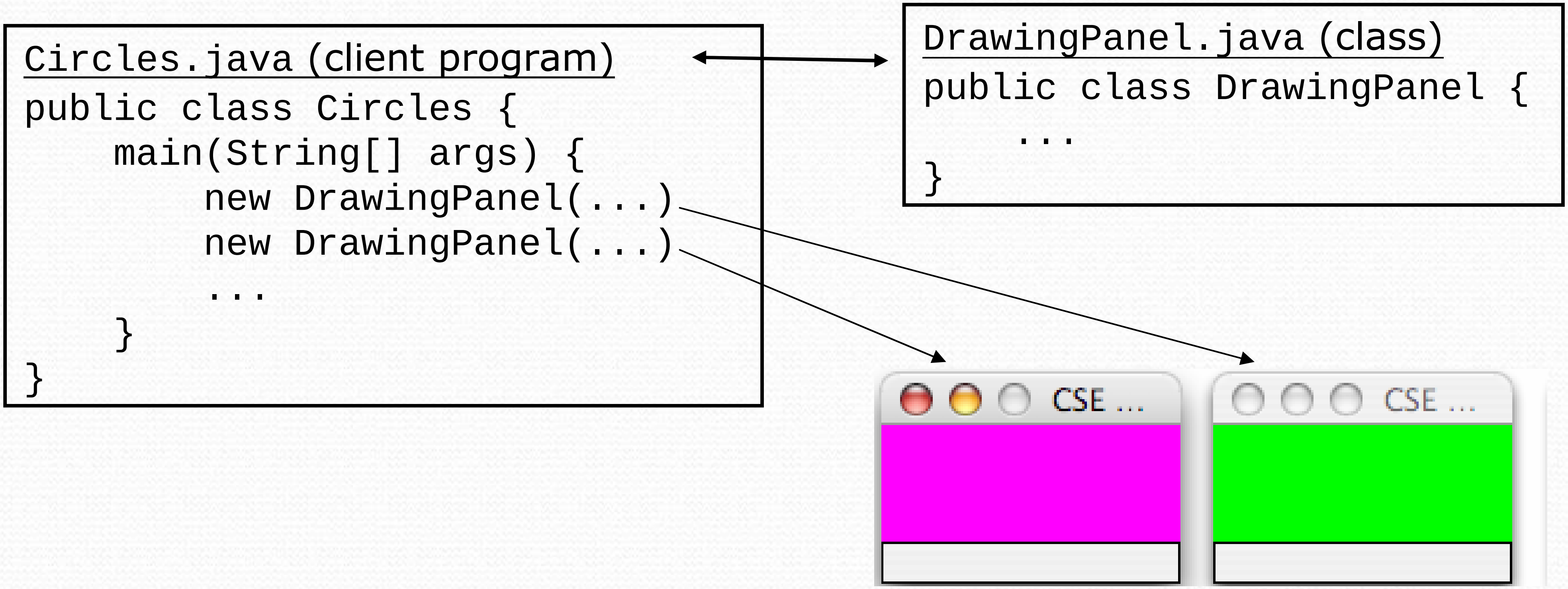
Observations

- This problem would be easier to solve if there were such a thing as a `Point` object.
  - A `Point` would store a city's x/y data.
  - We could compare distances between `Point`s to see whether the bomb hit a given city.
  - Each `Point` would know how to draw itself.
  - The overall program would be shorter and cleaner.



Clients of objects

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



Circles.java (client program)

```
public class Circles {
    main(String[] args) {
        new DrawingPanel(...)
        new DrawingPanel(...)
        ...
    }
}
```

DrawingPanel.java (class)

```
public class DrawingPanel {
    ...
}
```

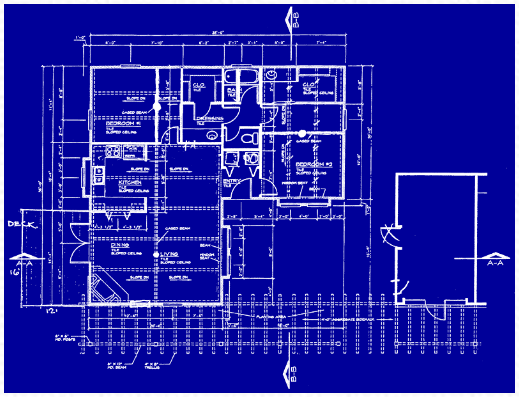
Classes and objects

- **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.
  - **object-oriented programming (OOP)**: Programs that perform their behavior as interactions between objects.

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 = "1,000,000 Miles"
volume = 17
battery life = 2.5 hrs

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

iPod #2

state:
song = "Letting You"
volume = 9
battery life = 3.41 hrs

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

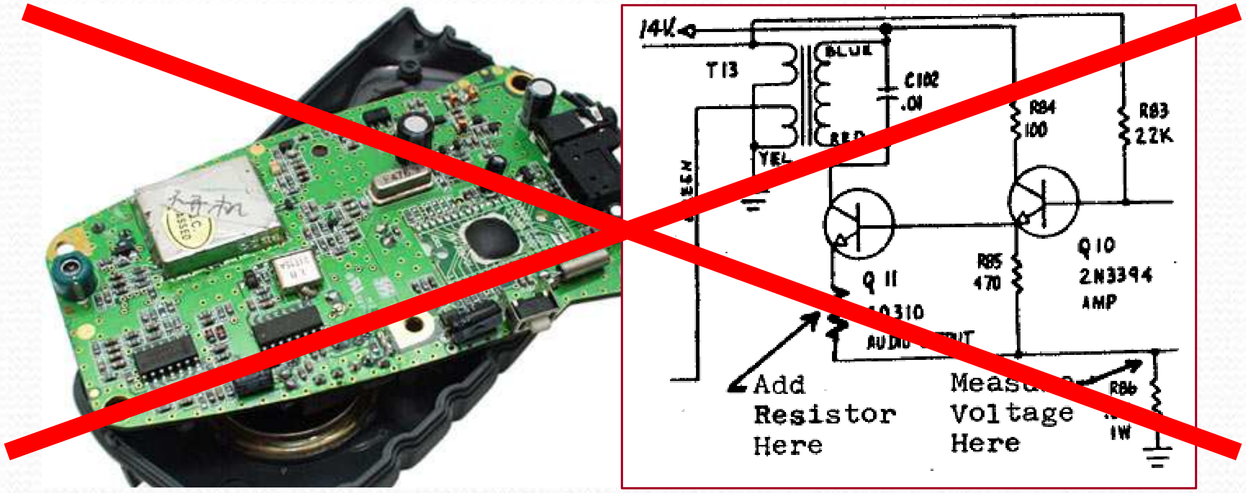
iPod #3

state:
song = "Discipline"
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.



Our task

- In the following slides, we will implement a `Point` class as a way of learning about classes.
  - We will 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 programs** will use the `Point` objects.

Point objects (desired)

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

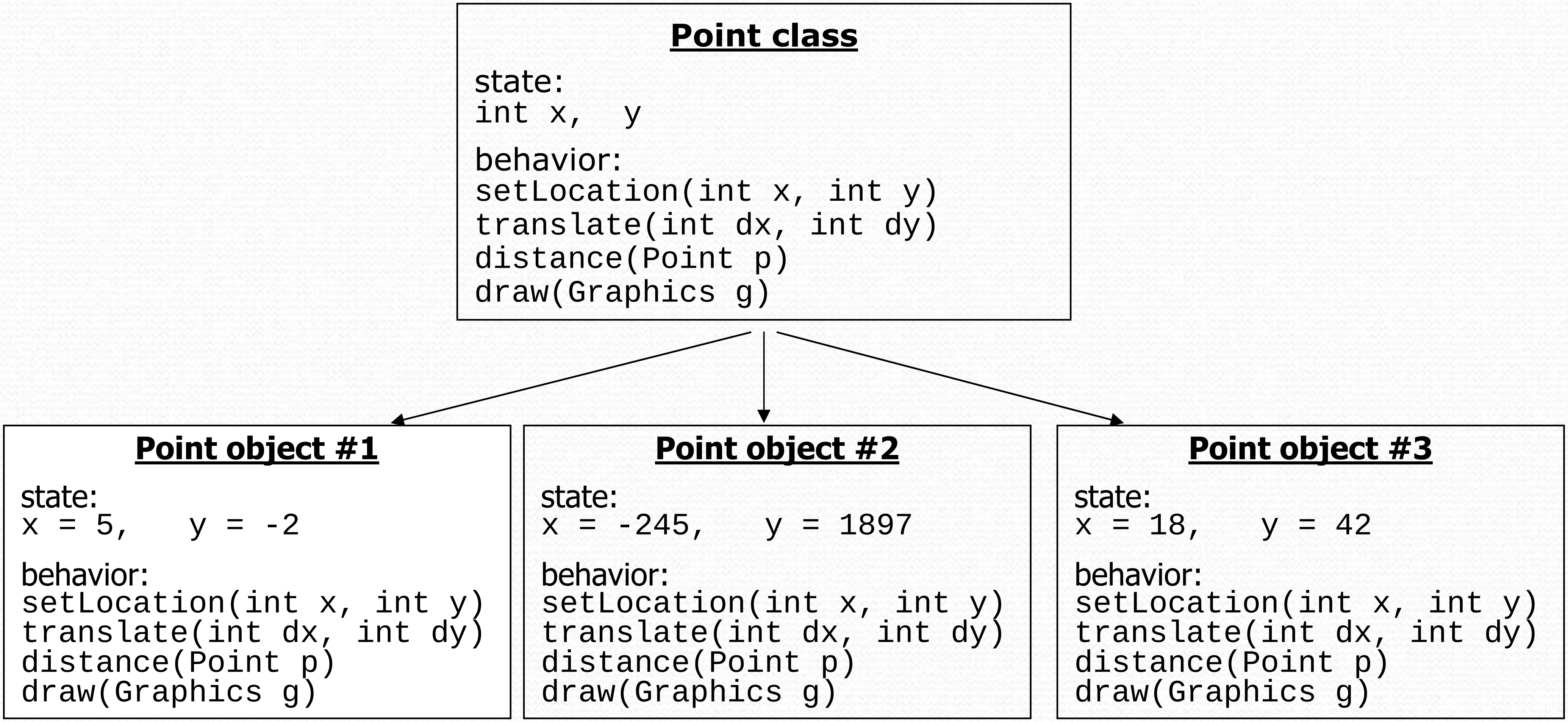
- Data in each `Point` object:

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) describes how to create objects.
- Each object contains its own data and methods.

Object state: Fields

reading: 8.2

self-check: #5-6

Point class, version 1

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

- Save this 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

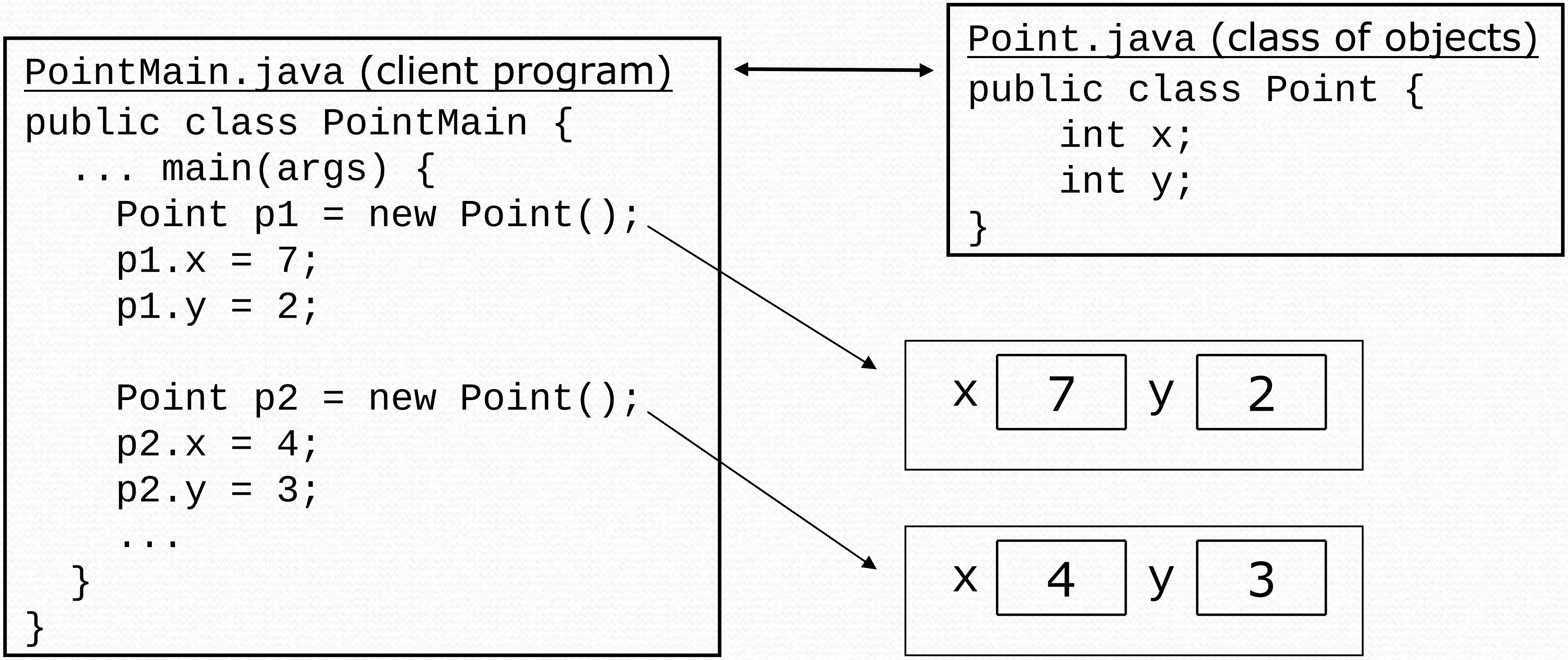
- Other classes can access/modify an object's fields.
  - access: **variable**.**field**
  - modify: **variable**.**field** = **value**;

- 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(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;
}
```

PointMain client example

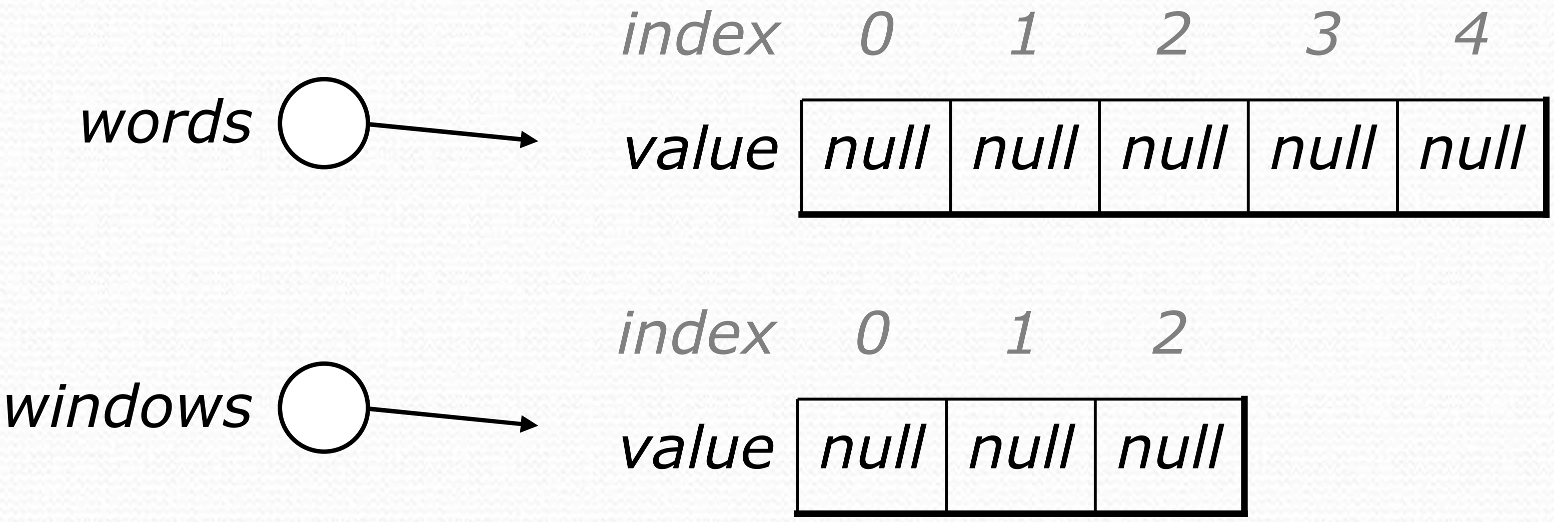
```
public class PointMain {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point();
        p1.y = 2;
        Point p2 = new Point();
        p2.x = 4;

        System.out.println(p1.x + "," + p1.y);   // 0,2

        // move p2 and then print it
        p2.x += 2;
        p2.y++;
        System.out.println(p2.x + "," + p2.y);   // 6,1
    }
}
```

- Exercise: Modify the Bomb program to use `Point` objects.

Arrays of objects

- **`null` :** A reference that does not refer to any object.
  - The elements of an array of objects are initialized to `null`.

```
String[] words = new String[5];
DrawingPanel[] windows = new DrawingPanel[3];
```



index	*0*	*1*	*2*	*3*	*4*
value	*null*	*null*	*null*	*null*	*null*

index	*0*	*1*	*2*
value	*null*	*null*	*null*

Things you can do w/ `null`

- store `null` in a variable or an array element

  ```
  String s = null;
  words[2] = null;
  ```

- print a `null` reference

  ```
  System.out.println(s);    // output: null
  ```

- ask whether a variable or array element is `null`

  ```
  if (words[i] == null) { ...
  ```

- pass `null` as a parameter to a method

- return `null` from a method (often to indicate failure)

Null pointer exception

- **dereference**: To access data or methods of an object with the dot notation, such as `s.length()`.
  - It is illegal to dereference `null` (causes an exception).
  - `null` is not any object, so it has no methods or data.

```
String[] words = new String[5];
System.out.println("word is: " + words[0]);
words[0] = words[0].toUpperCase();
```

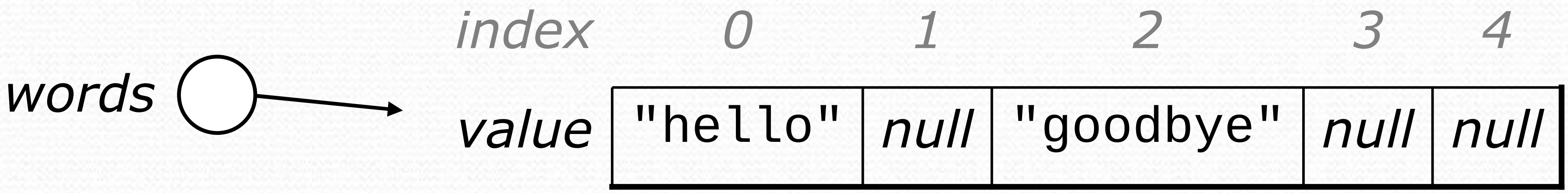
Output:

```
word is: null
Exception in thread "main"
java.lang.NullPointerException
        at Example.main(Example.java:8)
```

Looking before you leap

- You can check for `null` before calling an object's methods.

```
String[] words = new String[5];
words[0] = "hello";
words[2] = "goodbye";    // words[1], [3], [4] are null

for (int i = 0; i < words.length; i++) {
    if (words[i] != null) {
        words[i] = words[i].toUpperCase();
    }
}
```



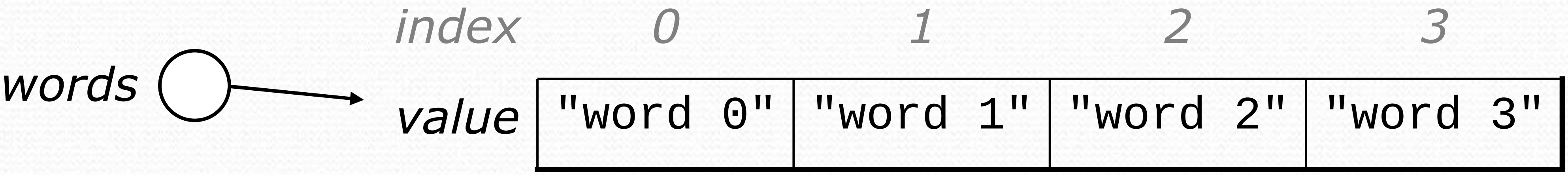
index	0	1	2	3	4
value	"hello"	null	"goodbye"	null	null

Two-phase initialization

1) initialize the array itself (each element is initially `null`)

2) initialize each element of the array to be a new object

```
String[] words = new String[4];            // phase 1
for (int i = 0; i < words.length; i++) {
    coords[i] = "word " + i;               // phase 2
}
```



index	0	1	2	3
value	"word 0"	"word 1"	"word 2"	"word 3"