

hi

# Building Java Programs

Chapter 8  
Lecture 8-1: Classes and Objects

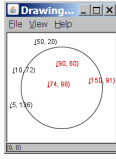
**reading: 8.1 – 8.3**

Copyright 2010 by Pearson Education

## 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? 100
Blast site y? 100
Blast radius? 75
Kaboom!
```

Copyright 2010 by Pearson Education

## A bad solution

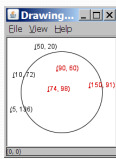
```
Scanner input = new Scanner(new File("cities.txt"));
int cityCount = input.nextInt();
int[] xCoords = new int[cityCount];
int[] yCoords = new int[cityCount];
for (int i = 0; i < cityCount; i++) {
    xCoords[i] = input.nextInt(); // read each city
    yCoords[i] = input.nextInt();
}
...
```

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

Copyright 2010 by Pearson Education

## Observations

- The data in this problem is a set of points.
- It would be better stored as `Point` objects.
  - A `Point` would store a city's x/y data.
  - We could compare distances between `Points` 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.



Copyright 2010 by Pearson Education

## Clients of objects

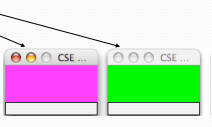
- client program:** A program that uses objects.
  - Example: `Bomb` is a client of `DrawingPanel` and `Graphics`.

**Bomb.java (client program)**

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

**DrawingPanel.java (class)**

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



Copyright 2010 by Pearson Education

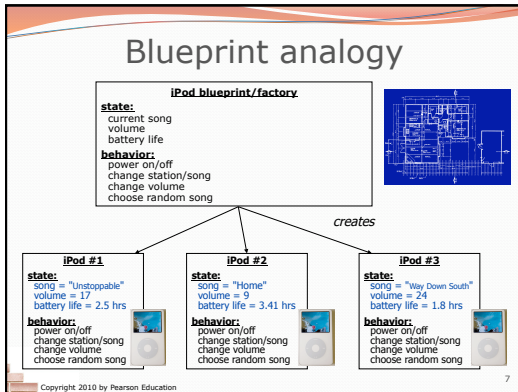
## Classes and objects

- class:** A program entity that represents either:
  - A program / module, or
  - 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.

Copyright 2010 by Pearson Education

bye

hi



### 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 defining 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)

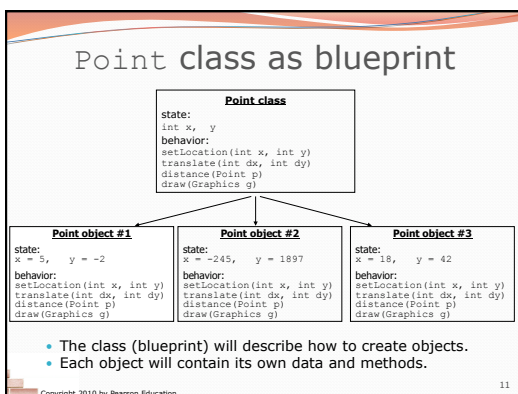
```
Point p1 = new Point(5, -2);
Point p2 = new Point();           // origin, (0, 0)
```

- 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



## Object state: Fields

**reading: 8.2**

bye

2

hi

### 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).

Copyright 2010 by Pearson Education 13

### 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  
}
```

Copyright 2010 by Pearson Education 14

### 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
```

Copyright 2010 by Pearson Education 15

### 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;  
}
```

Copyright 2010 by Pearson Education 16

### 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.

Copyright 2010 by Pearson Education 17

### Arrays of objects

- **null**: A value 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  
words value null null null null null
```

```
index 0 1 2  
windows value null null null
```

Copyright 2010 by Pearson Education 18

bye

3

hi

### 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); // null
```
- ask whether a variable or array element is null  

```
if (words[2] == null) { ...
```
- pass null as a parameter to a method  

```
System.out.println(null); // null
```
- return null from a method (often to indicate failure)  

```
return null;
```

Copyright 2010 by Pearson Education 19

### 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(); // ERROR
```

Output:

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

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

Copyright 2010 by Pearson Education 20

### 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();  
    }  
}
```

words

	index	0	1	2	3	4
value		"HELLO"	null	"GOODBYE"	null	null

Copyright 2010 by Pearson Education 21

### 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++) {  
    words[i] = "word" + i; // phase 2  
}
```

words

	index	0	1	2	3
value		"word0"	"word1"	"word2"	"word3"

Copyright 2010 by Pearson Education 22

### Bomb answer 1

```
import java.awt.*;  
import java.io.*;  
import java.util.*;  
  
// Displays a set of cities and simulates dropping a "bomb" on them.  
public class Bomb {  
    public static void main(String[] args) throws FileNotFoundException {  
        DrawingPanel panel = new DrawingPanel(200, 200);  
        Graphics g = panel.getGraphics();  
  
        Scanner input = new Scanner(new File("cities.txt"));  
        Point[] cities = readCities(input, g);  
  
        // drop the "bomb"  
        Scanner console = new Scanner(System.in);  
        Point bomb = new Point();  
        System.out.print("Blast site x? ");  
        bomb.x = console.nextInt();  
        System.out.print("Blast site y? ");  
        bomb.y = console.nextInt();  
        System.out.print("Blast radius? ");  
        int radius = console.nextInt();  
        boom(bomb, radius, cities, g);  
    }  
}
```

Copyright 2010 by Pearson Education 23

### Bomb answer 2

```
// Reads input file of cities and returns them as array of Points.  
public static Point[] readCities(Scanner input, Graphics g) {  
    int numCities = input.nextInt(); // first line = # of cities  
    Point[] cities = new Point[numCities];  
    for (int i = 0; i < cities.length; i++) {  
        cities[i] = new Point();  
        cities[i].x = input.nextInt(); // read city x/y from file  
        cities[i].y = input.nextInt();  
        g.fillOval(cities[i].x, cities[i].y, 3, 3);  
        g.drawString("(" + cities[i].x + ", " + cities[i].y + ")",  
            cities[i].x, cities[i].y);  
    }  
    return cities;  
}  
  
// Simulates dropping a bomb at the given location on the given cities.  
public static void boom(Point bomb, int radius, Point[] cities, Graphics g) {  
    g.setColor(Color.RED);  
    g.drawOval(bomb.x - radius, bomb.y - radius, 2 * radius, 2 * radius);  
    for (int i = 0; i < cities.length; i++) {  
        int dx = cities[i].x - bomb.x;  
        int dy = cities[i].y - bomb.y;  
        double distance = Math.sqrt(dx * dx + dy * dy);  
        if (distance <= radius) {  
            g.fillOval(cities[i].x, cities[i].y, 3, 3);  
            g.drawString("(" + cities[i].x + ", " + cities[i].y + ")",  
                cities[i].x, cities[i].y);  
        }  
    }  
    System.out.println("Kaboom!");  
}
```

Copyright 2010 by Pearson Education 24

bye

4