Building Java Programs

Chapter 7

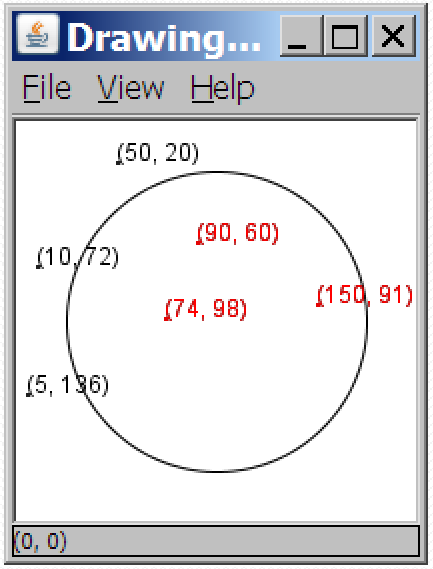
Lecture 27: More on ArrayList, Reference Semantics
Command Line Arguments

reading: 10.1, 4.3, 3.3, page 414

Recall: earthquake 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
```



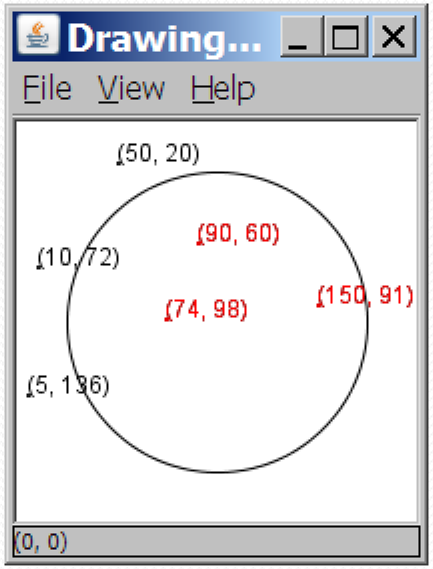
- Write a program to draw the cities on a `DrawingPanel`, then color the cities red that are within the radius of effect of the earthquake:

```
Epicenter x/y? 100 100
Radius of effect? 75
```

We made it a little too easy

- 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
```



- What if we didn't know the number of cities in advance?
  - Yes, "use a while loop", but first we have to make the array!

Solutions

1. Read the file twice
   - First time just to count how many cities
   - Okay, but won't work if the problem reads cities from the console until the user types, "STOP"
2. Resize the array as necessary
   - A common idiom worth knowing
   - Doubling each time is a very elegant policy
     - Never waste more than half the space
     - And you won't make very many arrays (2^{25} > 32 million)
3. Use an ArrayList
   - Just call `add` each time
   - Probably the ArrayList library is using something like resizing, *but that's somebody else's problem*
     - *Don't reinvent the wheel*

Parameters - value semantics

- Recall: Java parameters are initialized by *copying the value*
- The parameter is a different variable
- Assigning to a parameter variable has no effect on callers
- For arrays and objects, the parameter is a reference to the object. So the *reference* is copied (not the array or object itself).
  - Consequence: if you change the array or object in the called method, it's the same array or object that the caller has!

An int parameter

- Consider this example:

```
public static void main(String[] args) {
 int j = 8;
 octopus(j);
 octopus(j+2);
 System.out.println("in main - j = " + j);
}

public static void octopus(int k) {
    System.out.println("starting octopus - k = " + k);
    k = 20;
    System.out.println("leaving octopus - k = " + k);
}
```

Output:

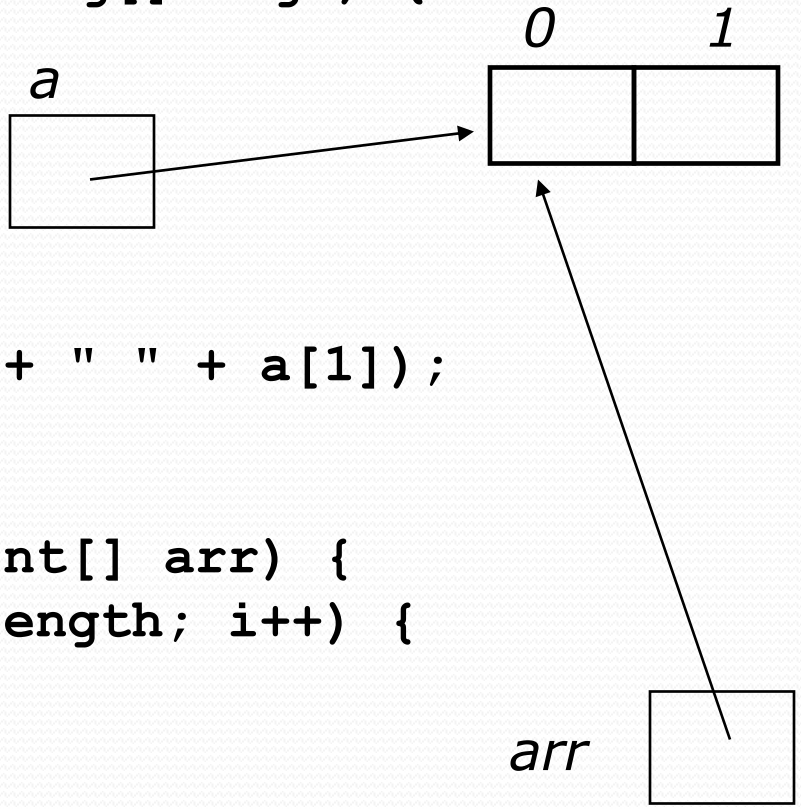
```
starting octopus - k = 8
leaving octopus - k = 20
starting octopus - k = 10
leaving octopus - k = 20
in main - j = 8
```

An array parameter

```
public static void main(String[] args) {
    int[] a = new int[2];
    a[0] = 42;
    a[1] = 64;
     tens(a);

    System.out.println(a[0] + " " + a[1]);
  }

  public static void tens(int[] arr) {
      for (int i=0; i<arr.length; i++) {
          arr[i] = 10;
      }
  }
```



An object parameter

```
public static void main(String[] args) {
 Point p = new Point(10,20);
 System.out.println("first p = " + p);
 m1(p);
 System.out.println("then p = " + p);
 m2(p);
 System.out.println("finally p = " + p);
}

public static void m1(Point q) {
    q.setLocation(100,100);
}

public static void m2(Point q) {
    q = new Point(200,200);
}
```

Output:

```
first p = (10, 20)
then p = (100, 100)
finally p = (100, 100)
```

Mini-exercise

- What does this print?

```
public static void main(String[] args) {
   Point p = new Point(10,20);
   Point q = new Point(50,50);
   Point r = p;
   r.setLocation(0,0);
   System.out.println("p = " + p);
   System.out.println("q = " + q);
   System.out.println("r = " + r);
}
```

Mini-exercise - answer

```
public static void main(String[] args) {
    Point p = new Point(10,20);
    Point q = new Point(50,50);
    Point r = p;
    r.setLocation(0,0);
    System.out.println("p = " + p);
    System.out.println("q = " + q);
    System.out.println("r = " + r);
}
```

Output:

```
p = (0, 0)
q = (50, 50)
r = (0, 0)
```

Mini-exercise #2

- What does this print?

```
public static void main(String[] args) {
    int[] a = new int[3];
    a[0] = 2;
    a[1] = 4;
    a[2] = 6;
    double(a);
    System.out.println(a[0] + " " + a[1] + " " + a[2]);
}

public static void double(int[] arr) {
     for (int i=0; i<arr.length; i++) {
       arr[i] = 2*arr[i];
     }
}
```

Mini-exercise #2 - answer

```
public static void main(String[] args) {
  int[] a = new int[3];
  a[0] = 2;
  a[1] = 4;
  a[2] = 6;
  double(a);
  System.out.println(a[0] + " " + a[1] + " " + a[2]);
}

public static void double(int[] arr) {
    for (int i=0; i<arr.length; i++) {
         arr[i] = 2*arr[i];
    }
}
```

- Output:

```
4 8 12
```

Mini-exercise #3

- What does this print?

```
public static void main(String[] args) {
  int[] a = new int[3];
  a[0] = 2;
  a[1] = 4;
  a[2] = 6;
  double(a);
  System.out.println(a[0] + " " + a[1] + " " + a[2]);
}

public static void double(int[] arr) {
   arr = new int[20];
   for (int i=0; i<arr.length; i++) {
     arr[i] = 2*arr[i];
   }
}
```

Mini-exercise #3 - answer

```
public static void main(String[] args) {
    int[] a = new int[3];
    a[0] = 2;
    a[1] = 4;
    a[2] = 6;
    double(a);
    System.out.println(a[0] + " " + a[1] + " " + a[2]);
}

public static void double(int[] arr) {
     arr = new int[20];
     for (int i=0; i<arr.length; i++) {
          arr[i] = 2*arr[i];
     }
}
```

- Output:

```
2 4 6
```

Command Line arguments

- You can pass arguments to the "main" method of a Java program
  - From jGRASP: pick the "build" menu item and check "run arguments". This opens an edit buffer for the command arguments just above the code editing page
  - When starting Java from a shell: just write the arguments after the word "java"
- These command line arguments are passed to the "main" method, in an array of Strings.

```
// print out the number of command line arguments and their values
public class CommandLineExample {
    public static void main(String[] args) {
        System.out.println("number of arguments: " + args.length);
        System.out.println("arguments: " + Arrays.toString(args));
    }

}
```

- Now, as promised in Lecture 1, we understand everything about “public static void main(String[] args)”