Building Java Programs

Chapter 8: Classes

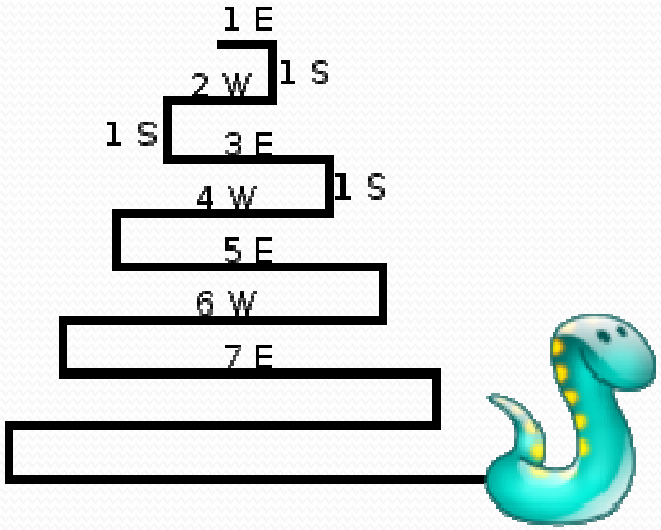
Lecture 8-3: More Critters, static

Testing Critters

- Focus on one specific Critter of one specific type
  - Only spawn 1 of each Critter type
- Make sure your fields update properly
  - Use `println` statements to see field values
- Look at the behavior one step at a time
  - Use "Step" rather than "Go"

A complex Critter: Snake

- Slithers in a wider and wider pattern
- ROAR 50% of the time; POUNCE 50% of the time
- Never hungry
- Displayed as an "S"
- Has a custom color



Determining necessary fields

- Information required to decide what move to make?
  - Direction to go in
  - Length of current cycle
  - Number of moves made in current cycle
- Information required to decide how to fight?
  - A `Random` object

Static fields and methods

Critter: Drunken Frat Guy

- All DFG Critters are trying to get to the same party
- The party is at a randomly-generated location
  - On a 60px wide by 50px tall world
- They stumble north then east until they reach the party

DFG: a flawed solution

```
import java.util.*;

public class DrunkenFratGuy extends Critter {
   private int partyX;
   private int partyY;

   public DrunkenFratGuy() {
         Random r = new Random();
         partyX = r.nextInt(60);
         partyY = r.nextInt(50);
   }

   public Direction getMove() {
         if(partyY != getY()) {
                  return Direction.NORTH;
         } else if(partyX != getX()) {
                  return Direction.EAST;
         } else {
                  return Direction.CENTER;
         }
   }
}
```

DFG: Where did they all go?

- Each DFG is heading to its own party!
- We need a way for Critters of a type to share information
- Tournament-winning Huskies do this
  - Hunt in packs
  - Don't kill each other
  - Share location of opponents

Static fields vs. fields

- **static**: Part of a class, rather than part of an object.
  - A single static field is shared by all objects of that class

- static field, general syntax:

```
private static <type> <name>;
```

or,

```
private static <type> <name> = <value>;
```

  - Example:

```
private static int count = 0;
```

Static field example

- Count the number of `Husky` objects created:

```
public class Husky implements Critter {
    // count of Huskies created so far
    private static int objectCount = 0;

    private int number;    // each Husky has a number

    public Husky() {
        objectCount++;
        number = objectCount;
    }

    ...

    public String toString() {
        return "I am Husky #" + number +
               "out of " + objectCount;
    }
}
```

Static methods

- **static method**: part of a class, not part of an object.
  - good places to put code related to a class, but not directly related to each object's state
  - shared by all objects of that class
  - does not understand the *implicit parameter*; therefore, cannot access fields directly
  - if `public`, can be called from inside or outside the class

- Declaration syntax: *(same as we have seen before)*

```
public static <return type> <name>(<params>) {
    <statements>;
}
```

Static method example 1

- Java's built-in `Math` class has code that looks like this:

```
public class Math {
    ...
    public static int abs(int a) {
        if (a >= 0) {
            return a;
        } else {
            return -a;
        }
    }

    public static int max(int a, int b) {
        if (a >= b) {
            return a;
        } else {
            return b;
        }
    }
}
```

Static method example 2

- Adding a static method to our `Point` class:

```
public class Point {
    ...
    // Converts a String such as "(5, -2)" to a Point.
    // Pre: s must be in valid format.

    public static Point parse(String s) {
        s = s.substring(1, s.length() - 1);  // "5, -2"
        s = s.replaceAll(",", "");           // "5 -2"

        // break apart the tokens, convert to ints
        Scanner scan = new Scanner(s);
        int x = scan.nextInt();              // 5
        int y = scan.nextInt();              // 2

        Point p = new Point(x, y);
        return p;
    }
}
```

Calling static methods, outside

- Static method call syntax (*outside* the class):

  ***<class name>*.*<method name>*(*<values>*);**

  - This is the syntax client code uses to call a static method.

  - Examples:

```
int absVal = Math.max(5, 7);

Point p3 = Point.parse("(-17, 52)");
```

Calling static methods, inside

- Static method call syntax (*inside* the class):

  <method name>(***<values>***);

  - This is the syntax the class uses to call its own static method.
  - Example:

```
public class Math {
    // other methods such as ceil, floor, abs, etc.
    // ...
    public static int round(double d) {
        if (d - (int) d >= 0.5) {
            return ceil(d);
        } else {
            return floor(d);
        }
    }
}
```

DFG: all go to the same party

```
import java.util.*;

public class DrunkenFratGuy extends Critter {
   private static int partyX;
   private static int partyY;

   public DrunkenFratGuy() {
         Random r = new Random();
         partyX = r.nextInt(60);
         partyY = r.nextInt(50);
   }

   public Direction getMove() {
         if(partyY != getY()) {
                  return Direction.NORTH;
         } else if(partyX != getX()) {
                  return Direction.EAST;
         } else {
                  return Direction.CENTER;
         }
   }
}
```