Building Java Programs

Chapter 8
Lecture 8-4: Static Methods and Data
Critter exercise: **Snake**

<table>
<thead>
<tr>
<th>Method</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td><code>public Snake()</code></td>
</tr>
<tr>
<td>eat</td>
<td>Never eats</td>
</tr>
<tr>
<td>fight</td>
<td>always forfeits</td>
</tr>
<tr>
<td>getColor</td>
<td>black</td>
</tr>
<tr>
<td>getMove</td>
<td><code>1 E, 1 S; 2 W, 1 S; 3 E, 1 S; 4 W, 1 S; 5 E, ...</code></td>
</tr>
<tr>
<td>toString</td>
<td>&quot;S&quot;</td>
</tr>
</tbody>
</table>
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

- Remembering things you've done in the past:
  - an `int` `counter`?
  - a `boolean` `flag`?
import java.awt.*; // for Color

public class Snake extends Critter {
    private int length;  // # steps in current horizontal cycle
    private int step;    // # of cycle's steps already taken

    public Snake() {
        length = 1;
        step = 0;
    }

    public Direction getMove() {
        step++;
        if (step > length) { // cycle was just completed
            length++;
            step = 0;
            return Direction.SOUTH;
        } else if (length % 2 == 1) {
            return Direction.EAST;
        } else {
            return Direction.WEST;
        }
    }

    public String toString() {
        return "S";
    }
}
Critter exercise: Student

- All the students are trying to get to the same party.
- The party is at a randomly-generated board location (On the 60-by-50 world).
- They stumble north then east until they reach the party.
A flawed solution

```java
import java.util.*; // for Random

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

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

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

- Problem: Each student goes to his own party. We want all students to share the same party location.
**Static members**

- **static**: Part of a class, rather than part of an object.
  - Object classes can have static methods *and fields*.
  - Not copied into each object; shared by all objects of that class.

```java
class
state:
private static int staticFieldA
private static String staticFieldB
behavior:
public static void someStaticMethodC()
public static void someStaticMethodD()
```

```java
object #1
state:
int field2
double field2
behavior:
public void method3()
public int method4()
public void method5()
```

```java
object #2
state:
int field1
double field2
behavior:
public void method3()
public int method4()
public void method5()
```

```java
object #3
state:
int field1
double field2
behavior:
public void method3()
public int method4()
public void method5()
```
Static fields

private static type name;
or,
private static type name = value;

- Example:
private static int theAnswer = 42;

- **static field**: Stored in the class instead of each object.
  - A "shared" global field that all objects can access and modify.
  - Like a class constant, except that its value can be changed.
Accessing static fields

- From inside the class where the field was declared:

  ```java
  fieldName
  fieldName = value;
  ```

  // get the value
  // set the value

- From another class (if the field is `public`):

  ```java
  ClassName.fieldName
  ClassName.fieldName = value;
  ```

  // get the value
  // set the value

- generally static fields are not `public` unless they are `final`

- Exercise: Modify the `BankAccount` class shown previously so that each account is automatically given a unique ID.

- Exercise: Write the working version of `Student`.
public class BankAccount {

    // static count of how many accounts are created
    // (only one count shared for the whole class)
    private static int objectCount = 0;

    // fields (replicated for each object)
    private String name;
    private int id;

    public BankAccount() {
        objectCount++;        // advance the id, and
        id = objectCount;     // give number to account
    }

    ...

    public int getID() {    // return this account's id
        return id;
    }
}

Copyright 2010 by Pearson Education
import java.util.*;  // for Random

public class Student extends Critter {
    // static fields (shared by all students)
    private static int partyX = -1;
    private static int partyY = -1;

    // object constructor/methods (replicated into each object)
    public Student() {
        if (partyX < 0 || partyY < 0) {
            Random r = new Random();  // the 1st one created
            partyX = r.nextInt(60);  // chooses the party location
            partyY = r.nextInt(50);  // for all students to go to
        }
    }

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

// the same syntax you've already used for methods
public static type name(parameters) {
    statements;
}

- **static method**: Stored in a class, not in an object.
  - Shared by all objects of the class, not replicated.
  - Does not have any *implicit parameter*, this;
    therefore, cannot access any particular object's fields.

- Exercise: Make it so that clients can find out how many *total* BankAccount objects have ever been created.
BankAccount solution

public class BankAccount {
    // static count of how many accounts are created
    // (only one count shared for the whole class)
    private static int objectCount = 0;

    // clients can call this to find out # accounts created
    public static int getNumAccounts() {
        return objectCount;
    }

    // fields (replicated for each object)
    private String name;
    private int id;

    public BankAccount() {
        objectCount++;
        // advance the id, and
        id = objectCount;
        // give number to account
    }

    ...

    public int getID() {
        // return this account's id
        return id;
    }
}
Advanced exercise

• A party is no fun if it's too crowded.

• Modify Student so that a party will be attended by no more than 10 students.
  
  • Every 10th student should choose a new party location for himself and the next 9 of his friends to be constructed.
    
    • first ten students go to party #1
    • next ten students go to party #2
    • ...

Copyright 2010 by Pearson Education
import java.util.*;  // for Random

public class Student extends Critter {
    // static fields (shared by all objects)
    private static int ourPartyX = -1;
    private static int ourPartyY = -1;
    private static int objectCount = 0;

    // chooses the party location for future students to go to
    public static void choosePartySpot() {
        Random r = new Random();
        ourPartyX = r.nextInt(60);
        ourPartyY = r.nextInt(50);
    }

    // object fields/constructor/methods (replicated in each object)
    private int myPartyX;
    private int myPartyY;

    ...
}
Advanced solution 2

... 

public Student() {
    // every 10th one chooses a new party spot for future students
    if (objectCount % 10 == 0) {
        choosePartySpot();
    }

    // must remember his party spot so they aren't all the same
    myPartyX = ourPartyX;
    myPartyY = ourPartyY;
}

public Direction getMove() {
    if (getY() != myPartyY) {
        return Direction.NORTH;
    } else if (getX() != myPartyX) {
        return Direction.EAST;
    } else {
        return Direction.CENTER;
    }
}


Multi-class systems

- Most large software systems consist of many classes.
  - One main class runs and calls methods of the others.

- Advantages:
  - code reuse
  - splits up the program logic into manageable chunks
Redundant program 1

```java
// This program sees whether some interesting numbers are prime.
public class Primes1 {
    public static void main(String[] args) {
        int[] nums = {1234517, 859501, 53, 142};
        for (int i = 0; i < nums.length; i++) {
            if (isPrime(nums[i])) {
                System.out.println(nums[i] + " is prime");
            }
        }
    }
}

// Returns the number of factors of the given integer.
public static int countFactors(int number) {
    int count = 0;
    for (int i = 1; i <= number; i++) {
        if (number % i == 0) {
            count++;
        } // i is a factor of the number
    }
    return count;
}

// Returns true if the given number is prime.
public static boolean isPrime(int number) {
    return countFactors(number) == 2;
}
```

Copyright 2010 by Pearson Education
// This program prints all prime numbers up to a maximum.
public class Primes2 {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Max number? ");
        int max = console.nextInt();
        for (int i = 2; i <= max; i++) {
            if (isPrime(i)) {
                System.out.print(i + " ");
            }
        }
        System.out.println();
    }
// Returns true if the given number is prime.
    public static boolean isPrime(int number) {
        return countFactors(number) == 2;
    }
// Returns the number of factors of the given integer.
    public static int countFactors(int number) {
        int count = 0;
        for (int i = 1; i <= number; i++) {
            if (number % i == 0) {
                count++;
            } // i is a factor of the number
        }
        return count;
    }
}
Classes as modules

- **module**: A reusable piece of software, stored as a class.
- **Example module classes**: Math, Arrays, System

```java
// This class is a module that contains useful methods
// related to factors and prime numbers.
public class Factors {
    // Returns the number of factors of the given integer.
    public static int countFactors(int number) {
        int count = 0;
        for (int i = 1; i <= number; i++) {
            if (number % i == 0) {
                count++;
                // i is a factor of the number
            }
        }
        return count;
    }

    // Returns true if the given number is prime.
    public static boolean isPrime(int number) {
        return countFactors(number) == 2;
    }
}
```
More about modules

- A module is a partial program, not a complete program.
  - It does not have a `main`. You don't run it directly.
  - Modules are meant to be utilized by other `client` classes.

- Syntax:

  ```
  class . method ( parameters ) ;
  ```

- Example:

  ```
  int factorsOf24 = Factors . countFactors ( 24 ) ;
  ```
Using a module

// This program sees whether some interesting numbers are prime.
public class Primes {
    public static void main(String[] args) {
        int[] nums = {1234517, 859501, 53, 142};
        for (int i = 0; i < nums.length; i++) {
            if (Factors.isPrime(nums[i])) {
                System.out.println(nums[i] + " is prime");
            }
        }
    }
}

// This program prints all prime numbers up to a given maximum.
public class Primes2 {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Max number? ");
        int max = console.nextInt();
        for (int i = 2; i <= max; i++) {
            if (Factors.isPrime(i)) {
                System.out.print(i + " ");
            }
        }
        System.out.println();
    }
}
// Java's built in Math class is a module
public class Math {
    public static final double PI = 3.14159265358979323846;

    ...

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

    public static double toDegrees(double radians) {
        return radians * 180 / PI;
    }
}
A class is used for any of the following in a large program:

- a program: Has a main and perhaps other static methods.
  - example: GuessingGame, Birthday, MadLibs, CritterMain
  - does not usually declare any static fields (except final)

- an object class: Defines a new type of objects.
  - example: Point, BankAccount, Date, Critter, Student
  - declares object fields, constructor(s), and methods
  - might declare static fields or methods, but these are less of a focus
  - should be encapsulated (all fields and static fields private)

- a module: Utility code implemented as static methods.
  - example: Math