Class Relationships

November 3, 2006

Example: Sudoku_Solver.java

Solving Sudoku Puzzles With Recursion
(http://www.websudoku.com/)

Rule #1: 1..9 must be in each row

Sample that satisfies rule #1

Rule #2: 1..9 must be in each column
Sample that satisfies rules #1 and #2

Rule #3: 1..9 must be in each 3x3 window

Sample that satisfies rules #1, #2, and #3

Quick Review of Last Lecture

Visibility Modifiers

<table>
<thead>
<tr>
<th>Variables</th>
<th>public</th>
<th>private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Violate encapsulation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enforce encapsulation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Methods      | provide services to clients | support other methods in the class |

The static Modifier

- We declare static methods and variables using the static modifier
- It associates the method or variable with the class rather than with an object of that class
- Static methods are sometimes called class methods and static variables are sometimes called class variables
Static Variables

- Normally, each object has its own data space, but if a variable is declared as static, only one copy of the variable exists
  
  ```
  private static float price;
  ```
- Memory space for a static variable is created when the class is first referenced
- All objects instantiated from the class share its static variables
- Changing the value of a static variable in one object changes it for all others

Classes

- A class can contain data declarations and method declarations

```
int size;
int weight;
```

Objects – instances of classes

```
obj1
int size = 5;
int weight = 170;
```
```
obj2
int size = 10;
int weight = 130;
```

Note that the variables can have different values in the two objects

```
obj1
int size = 5;
int weight = 170;
```
```
obj2
int size = 10;
int weight = 130;
```

Objects – instances of a class with a static variable ‘size’

```
obj1
static int size;
```
```
obj2
static int size;
```

```
obj1
int weight = 170;
```
```
obj2
int weight = 130;
```

Changes if we declare a static variable

```
static int size;
int weight;
```
Objects – instances of classes

• Note that the variables can have different values in the two objects

```
int size = 5;
int weight = 170;
```

```
int size = 10;
int weight = 130;
```

Static Class Members

• The order of the modifiers can be interchanged, but by convention visibility modifiers come first

• Recall that the main method is static – it is invoked by the Java interpreter without creating an object

• Static methods cannot reference instance variables because instance variables don’t exist until an object exists

• However, a static method can reference static variables or local variables

Method Control Flow

• If the called method is in the same class, only the method name is needed

```
myMethod();
```

Accessing Variables

• If the called method is in the same class, only the method name is needed

```
myMethod();
myMethod compute
```

```
myVariable = 5;
```

OK

Accessing Variables

• Static methods cannot use non static class variables.

```
myMethod();
myMethod
```

```
myVariable = 5;
```

Error

```
int myVariable;
```

static

int myVariable;

OK

Accessing Variables

• Static methods can use static class variables

```
compute
```

```
myMethod;
```

```
static myMethod
```

```
myVariable = 5;
```

static int myVariable;

OK
**Static Class Members**

- Recall that a static method is one that can be invoked through its class name.
- For example, the methods of the `Math` class are static:
  ```java
  result = Math.sqrt(25);
  ```
- Variables can be static as well.
- Determining if a method or variable should be static is an important design decision.

**Static Methods**

```java
class Helper {
    public static int cube (int num) {
        return num * num * num;
    }
}
```

Because it is declared as static, the method can be invoked as:
```java
value = Helper.cube(5);
```

**Method Control Flow**

Static methods can only call other static methods within the same class.

**Class Relationships**

- Classes in a software system can have various types of relationships to each other.
- Three of the most common relationships:
  - Dependency: A uses B
  - Aggregation: A has-a B
  - Inheritance: A is-a B

**Dependency**

- A dependency exists when one class relies on another in some way, usually by invoking the methods of the other.
- We’ve seen dependencies in many previous examples.
- We don’t want numerous or complex dependencies among classes.
- Nor do we want complex classes that don’t depend on others.
- A good design strikes the right balance.
Dependancy Example: Client-Server

- Some dependencies occur between objects of the same class
- A method of the class may accept an object of the same class as a parameter
- For example, the `concat` method of the `String` class takes as a parameter another `String` object
  
  ```java
  str3 = str1.concat(str2);
  ```
- This drives home the idea that the service is being requested from a particular object

Concatenation Example

- The following example defines a class called `Rational` to represent a rational number
- A rational number is a value that can be represented as the ratio of two integers
- Some methods of the `Rational` class accept another `Rational` object as a parameter
- See `RationalTester.java` (page 297)
- See `Rational.java` (page 299)

Aggregation

- An aggregate is an object that is made up of other objects
- Therefore aggregation is a has-a relationship
  - A car has a chassis
  - A student has an address

Aggregation Example: Components of a Car
Aggregation

- In software, an aggregate object contains references to other objects as instance data

- The aggregate object is defined in part by the objects that make it up

- This is a special kind of dependency – the aggregate usually relies on the objects that compose it
Aggregation Example: Components of a Student

Student

First Name

Last Name

Home Address

School Address

John

Smith

21 Jump Street

800 Lancaster Ave.

Marsha

Jones

123 Main Street

800 Lancaster Ave.

Aggregation in UML

StudentBody

+ main (args : String[]): void

+ toString() : String

Student

- firstName : String

- lastName : String

- homeAddress : Address

- schoolAddress : Address

+ toString() : String

Address

- streetAddress : String

- city : String

- state : String

- zipCode : long

+ toString() : String

Aggregation

- In the following example, a Student object is composed, in part, of Address objects

- A student has an address (in fact each student has two addresses)

- See StudentBody.java (page 304)

- See Student.java (page 306)

- See Address.java (page 307)

- An aggregation association is shown in a UML class diagram using an open diamond at the aggregate end
THE END