CSE1710

Week 09, Lecture 16

Fall 2013 ◆ Tuesday, Nov 05, 2013

YORK UNIVERSITÉ UNIVERSITY

Big Picture

We are in week09. The remaining weeks are week10 - week13.

We will finish covering Chapter 4 this week.

On Tuesday, Nov 12, we will have a term test that **focuses on Ch 4**.

After this we move on to Ch 5 & 6, with a focus on images and strings.

YORK UNIVERSITÉ UNIVERSITY

Big Picture

Reading that was assigned…

- ❑ re-read section 4.2 “The Life of an Object” pp. 136-148, with a focus on 4.2.4, 4.2.5, 4.2.6
- ❑ read section 4.3 “The Object’s State” pp. 149-157
- ❑ review Ch 4 KC’s 11-16
- ❑ do Ch 4 RQ’s 23-34
- ❑ do Ch 4 Ex’s 4.12-4.22

YORK UNIVERSITÉ UNIVERSITY

Skills you should have…

- at runtime, be able to state how many unique objects exist in memory; be able to describe what is happening to these objects as a function of method invocations and/or field modification.
- describe the function of assigning an object reference to `null`
- be able to describe the difference between == and the `equals` method
- explain the idea of obligatory methods and be able to identify them (from memory and/or an API)

YORK UNIVERSITÉ UNIVERSITY

Skills you should have…

- understand the process of garbage collection and be able predict the results of garbage collection on the contents of memory
- understand what accessor and mutator methods are; be able to distinguish their function from the alternative method of direct access of fields
- understand the notion of legal state for objects, describe the importance of maintaining legal state;
  - describe how mutators enforce legality, whereas public visibility of attributes cannot.
  - understand how this implements the core principle of encapsulation.

Skills you should have…

- understand the difference between static methods and static attributes; understand how static attributes operate
- understand the characteristics of object and/or class attributes that are final
- understand how `final` and `static` can be combined for class attributes

Textbook Exercise (riff on pp.145-146)

- At runtime,
  - how many references will be created?
  - how many objects will be created?
  - do any objects have the same state?
  - predict the output

```
Fraction f1 = new Fraction(3, 5);
Fraction f1 = f2;
Fraction f3 = new Fraction(2, 7);
Fraction f4 = new Fraction(6, 10);
Fraction f5 = f4;
output.printf(“f1==f2, result: %s%n”, f1==f2);
output.printf(“f4==f5, result: %s%n”, f4==f5);
output.printf(”f1==f4, result: %s%n”, f1 == f4);
output.printf(”f1.equals(f4), result: %s%n”, f1.equals(f4));
```

Exercises 4.1-4.10

- consult the API for class `type.lib.Item`
  - we will revisit this class in this week’s lab exercises

The class `Stock`

- We will use the `Stock` class from type.jar for this example
- A *public* company is a company that offers its stock/ shares for sale to the general public, typically through a stock exchange
- A public company has a full name and is represented by a two-character symbol
  - e.g., name: “Alpha Bravo Co.”, symbol: “.AB”
- At any given point in time, the company’s shares have a **selling price**.
- We use the class `Stock` to encapsulate a single share

The class `Stock`

- When constructing a `Stock` instance, the client must specify the two-character symbol.
- The `Stock` class’ `getName()` accesses the name of the company that corresponds to the stock’s two-character stock exchange symbol:

```
ALPHA of BRAVO Company
Alpha of Bravo Company
```

- Whether the name is upper-case or camel-case, this is determined by the boolean flag `titleCaseName`
- The attribute is **public** and **static**

The class `Stock`

- The `Stock` class' `toString()` produces a "nice" string representation consisting of something like:

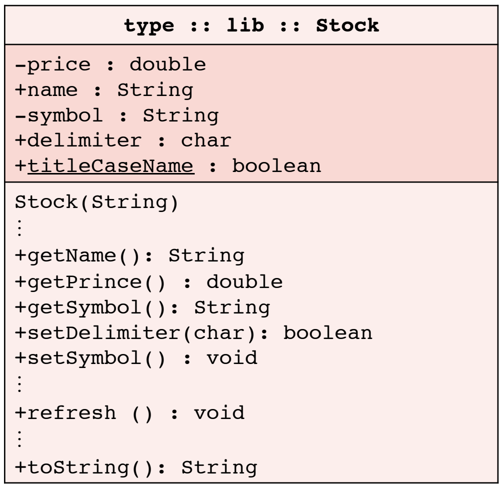
```
.AB*ALPHA of BRAVO Company
.AB:ALPHA of BRAVO Company
.AB+ALPHA of BRAVO Company
.AB ALPHA of BRAVO Company
.AB#ALPHA of BRAVO Company
.AB.ALPHA of BRAVO Company
```

- The character is red is called the delimiter
- The client can specify the character to be used for this delimiter

The class `Stock`

- The `Stock` class' `getPrice()` retrieves the most-recently fetched version of the price. Upon instantiation, the current price is fetched.
- The method `refresh()` will connect to the Stock Exchange server and fetch the current version of the price

UML Diagram



```
type :: lib :: Stock
-price : double
+name : String
-symbol : String
+delimiter : char
+titleCaseName : boolean
Stock(String)
:
+getName(): String
+getPrince() : double
+getSymbol(): String
+setDelimiter(char): boolean
+setSymbol() : void
:
+refresh () : void
:
+toString(): String
```

Exercise

- At runtime,
  - how many references will be created?
  - how many objects will be created?
  - do any objects have the same state?
  - predict the output

```
Stock s1 = new Stock(".AB");
Stock s2 = new Stock(".BT");
Stock s3 = new Stock(".XY");
Stock s4 = new Stock(".AB");
output.printf("s1: %s%n", s1.toString());
output.printf("s2: %s%n", s2.toString());
output.printf("s3: %s%n", s3.toString());
output.printf("s1 == s4: %s%n", s1==s4);
output.printf("s1.equals(s4): %s%n", s1.equals(s4));
```

Exercises 4.11-4.12

- consult the API for class `type.lib.Stock`
    - we will revisit this class in this week's lab exercises