DEPAUL UNIVERSITY
COLLEGE OF COMPUTING AND DIGITAL MEDIA

CSC 281 NOTES
Spring 2019: Amber Settle

Week 5

Announcements

- Discuss the third assignment
- The fourth assignment is due tonight at 9 pm – questions?
- The fifth assignment is due Wednesday, May 1st at 9 pm

Object-oriented programming in Java

When we define a class in Java, we describe:

1. What **properties** will make up the state of the object from the class
2. What **behaviors** the object in the class will be capable of performing

The properties and behaviors are what are called the **members or fields** of the class.

1. Properties are given by data members or **instance variables**
2. Behaviors are given by **methods** or instance methods

Typically, instance variables are not directly accessible to a program using an object from a class (i.e. private). The instance methods are the only access to the outside world (i.e. public).

This idea is known as **encapsulation**. You use objects from a class via public methods without knowing how the interior of the class is implemented.

Static members

Instance variables and methods of the classes we have written are **properties and behaviors of the individual objects** we declare from the class.

To refer to them, we use **objectName.memberName**.

Example: firstOne.sell(10);

Static members (or class members) belong to the class as a whole, rather than to any particular object of the class.

To refer to them, we use **className.memberName**.

Example: Item.reportTotalValue();

We do not have to declare an object from these classes to use these methods.

Class variables and methods can only be allowed to access class variables and other class methods. On the other hand, instance methods can access both class and instance variables and methods.

Example: Add a static member to the **Item** class to represent the total inventory value of the warehouse. Modify the instance methods to keep track of the new member correctly.

Rational number class

A **rational number** consists of a **numerator** and **denominator**. Both are integers. The denominator must be non-zero. These will be the instance variables for the class.

The **methods** for this class:

1. public Rational(int, int) – Initialize the numerator and denominator of a new Rational
2. public Rational(int) – Initialize the numerator and set the denominator to 1
3. public Rational add(Rational) – Add two Rationals
4. public Rational multiply(Rational) – Multiply two Rationals
5. public boolean equals(Rational) – Check if two Rationals are the same

Note: The binary operators (add, multiply) take only one argument. Why?

Implementation of the class

At this point, we have enough information to write a simple driver class for the Rational class.

Create the file **UseRationals.java**. The program creates two rationals, adds, multiplies, and compares them.

In order for this driver program to run, we need to have the Rational class written. Create that class in the file **Rational.java**.

Note: Since some of these methods have objects as their parameters, we must create new objects within these methods' bodies.

Improvement on the class: **Reduce to lowest terms**

Example: Our two objects are not equal, although they both represent the same value.

Next improvement: Modify the class to handle **negative rationals**.

How should the negative value be represented? What methods need to change to make the class work correctly?

Final change: Implement a static variable to keep track of the number of rationals created.

Add a static method to return the variable. And finally modify all the methods that have to update the variable.