CSE3040 Java Language

Lecture 09: Object-Oriented Programming (3)

Dept. of Computer Engineering,
Sogang University

Four Principles of Object-Oriented Programming

- Encapsulation
  - hide internal implementation by restricting access to public methods
  - instance variables and some methods are kept private

- Abstraction
  - use of "Interface", a specification without implementation
  - abstract classes

- Inheritance
  - "is-a" and/or "has-a" relationship between two objects
  - super class (parent class) vs. sub class (child class)
  - reuse the code of existing super classes

- Polymorphism
  - one name can have many different forms
  - static polymorphism: method overloading
  - dynamic polymorphism: method overriding

Interface

- Suppose we want to make a program where we average the first **n** numbers in an integer sequence.
- In the program, we write a static method (**average**) that returns average of an integer sequence.
  - The first argument of the method is a class we are going to define (IntSequence).

```
public static double average(IntSequence seq, int n)
```

- One way to make IntSequence is to define it as class.

```
class IntSequence {
    private int i;

    public boolean hasNext() {
      return true;
    }
    public int next() {
      i++;
      return i;
    }
}
```

Interface

- Using class IntSequence, we can implement the **average** method.

```
public static double average(IntSequence seq, int n) {
    int count = 0;
    double sum = 0;
    while(seq.hasNext() && count < n) {
        count++;
        sum += seq.next();
    }
    return count == 0 ? 0 : sum / count;
}
```

Interface

- Suppose we have various types of integer sequences.
  - simple sequence: 1, 2, 3, 4, 5, ...
  - square sequence: 1, 4, 9, 16, 25, ...
  - digit sequence: sequence of digits in a number
    - e.g.) a digit sequence of 1729 is 1, 7, 2, 9.

- If we create different classes for each sequence, we need to have as many methods as the classes
  - SimpleSequence: public static double average(SimpleSequence seq, int n)
  - SquareSequence: public static double average(SquareSequence seq, int n)
  - DigitSequence: public static double average(DigitSequence seq, int n)

- Instead of creating a method for each class, we can use interface.

Interface

- An interface is a way to achieve abstraction in Java.
- We define an interface, which includes a group of methods.

```
interface IntSequence {
    boolean hasNext();
    int next();
}
```

  - This interface only has specifications (what methods are available), and does not have actual implementations. So an interface is an abstract concept.
  - All methods specified in an interface are public by default.
    - You can also include the keyword public.
  - A method without an implementation is called an abstract method.
    - You can also include the keyword abstract.

```
interface IntSequence {
    public abstract boolean hasNext();
    public abstract int next();
}
```

Interface

- Now, in order to actually use the interface, we need a class that implements the interface. - class SquareSequence must implement all methods in the interface IntSequence.

```
class SquareSequence implements IntSequence {
    private int i;

    public boolean hasNext() {
        return true;
    }
    public int next() {
        i++;
        return i*i;
    }
}
```

Interface

- Remember that we implemented a static method **average**.

```
public static double average(IntSequence seq, int n) {
    int count = 0;
    double sum = 0;
    while(seq.hasNext() && count < n) {
        count++;
        sum += seq.next();
    }
    return count == 0 ? 0 : sum / count;
}
```

- Now, we can call this method like this.

```
SquareSequence squares = new SquareSequence();
double avg = average(squares, 100);
```

  - Note that "squares" is of type SquareSequence.
  - Since SquareSequence implements interface IntSequence, it is possible to use the variable as an argument to the method.

Interface

- We can define another class that implements IntSequence.
  - class DigitSequence implements hasNext() and next().
  - class DigitSequence has an additional method (rest) that is not in the interface.

```
class DigitSequence implements IntSequence {
    private int number;
    public DigitSequence(int n) { number = n; }
    public boolean hasNext() { return number != 0; }
    public int next() {
        int result = number % 10;
        number /= 10;
        return result;
    }
    public int rest() { return number; }
}
```

- We can use the same method average with DigitSequence too.

```
DigitSequence digits = new DigitSequence(2345);
double avg = average(digits, 100);
```

Interface

- When we create an object of class DigitSequence, we would normally do this:

```
DigitSequence digits = new DigitSequence();
double avg = average(digits, 100);
```

- But, we can also define the variable as type of the interface, and then create an object of class DigitSequence.

```
IntSequence digits = new DigitSequence();
double avg = average(digits, 100);
```

- When class DigitSequence implements interface IntSequence,
  - IntSequence is a supertype of DigitSequence
  - DigitSequence is a subtype of IntSequence
- A supertype variable can be assigned with a subtype object.

```
IntSequence digits = new DigitSequence();
```

- You can declare an interface type variable, but you cannot create an instance of an interface.

```
IntSequence seq = new IntSequence();  // impossible
```

Interface

- If we want to assign a supertype variable to a subtype variable, then we need explicit type casting.

```
IntSequence sequence = new DigitSequence(2345);
DigitSequence digits = (DigitSequence)sequence;
System.out.println(digits.rest());
```

  - The casting works only when the casted variable actually refers to an object of DigitSequence (or its supertype.)

```
IntSequence sequence = new SquareSequence();
DigitSequence digits = (DigitSequence)sequence;
System.out.println(digits.rest());
```

  - This will lead to runtime exception called ClassCastException (exceptions will be discussed later)

Interface

- To avoid ClassCastException, we can use the operator instanceof to check the type of an object. - *object* instanceof *type* is true if *object* is subtype of *type*.

```
IntSequence sequence = new DigitSequence(2345);
if (sequence instanceof DigitSequence) {                // true
    DigitSequence digits = (DigitSequence)sequence;
    System.out.println(digits.rest());
}
```

```
IntSequence sequence = new SquareSequence();
if (sequence instanceof DigitSequence) {                // false
    DigitSequence digits = (DigitSequence)sequence;
    System.out.println(digits.rest());
}
```

```
IntSequence sequence = new SquareSequence();
if (sequence instanceof IntSequence) {                  // true
    DigitSequence digits = (DigitSequence)sequence;     // ClassCastException
    System.out.println(digits.rest());
}
```

```
DigitSequence sequence = new DigitSequence(2345);
if (sequence instanceof IntSequence) {                  // true
    DigitSequence digits = (DigitSequence)sequence;
    System.out.println(digits.rest());
}
```

Interface

- We can define an interface by extending another interface.

```
interface Closeable {
    void close();
}
```

```
interface Channel extends Closeable {
    boolean isOpen();
}
```

  - Then, any class that implements interface Channel must implement both methods close() and isOpen().

Interface

- A class may implement multiple interfaces.

```
public class FileSequence implements IntSequence, Closeable {
...
}
```

  - The class must implement all methods specified in IntSequence as well as all methods in Closeable.

Programming Lab #09

09-01. Using Interface

- Write a Java program that satisfies the following requirements.
  - Define a static method average that returns average of the first n numbers of an integer sequence.
  - Define class SimpleSequence, class SquareSequence, and class DigitSequence.
    - SimpleSequence generates an integer sequence such as 1, 2, 3, 4, 5, 6, 7, ...
    - SquareSequence generates a sequence of squares such as 1, 4, 9, 16, 25, 36, ...
    - DigitSequence generates a sequence of digits from a given integer in the reverse order. If the given number is 1527, the digit sequence is 7, 2, 5, 1.
  - Each class should implement the following methods:
    - boolean hasNext(): returns true if the next number exists in the sequence.
    - int next(): returns the next number in the sequence.
- Implement two versions of the program
  - One program does not use interface, and the other program uses interface

09-01. Using Interface

- An example implementation of class DigitSequence

```
class DigitSequence {
    private int number;
    public DigitSequence(int n) { number = n; }
    public boolean hasNext() { return number != 0; }
    public int next() {
        int result = number % 10;
        number /= 10;
        return result;
    }
    public int rest() { return number; }
}
```

End of Class

Instructor office: AS818A

Email: jso1@sogang.ac.kr