Lab 3

Chapter 3

Advanced Object-Oriented Concepts

Contents

- Lab1 : OneRowNim 게임 설계 및 구현
- Lab2 : 예외처리 실습

- Homework#2 :
  - BankAccount 클래스 및 예외처리 구현

[LAB1] CASE STUDY: Simulating a Two-Person Game

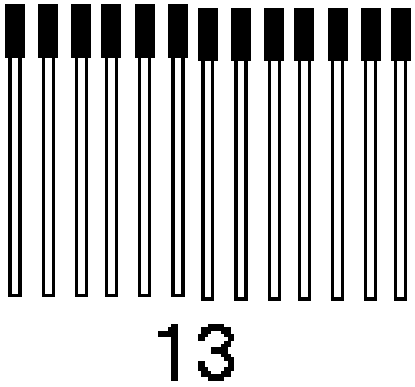
- Refer to …
  - http://turing.cs.trincoll.edu/~ram/cpsc115

Case Study: A Two-Person Game

- Design Steps
  - Problem Specification
  - Problem Decomposition
  - Class Design: OneRowNim
  - Method Decomposition

[Ref] Nim Game

- The Nim Game is a game for two players.
  - 성냥개비 같은 것을 쌓아 놓고 차례대로 몇 개씩 덜어내기를 해서 누가 맨 마지막 것을 집게 되는지를 겨루는 게임
- It consists of a row of 13 matches. Two players take alternately 1, 2 or 3 matches. The one, who takes the last match, wins.



- Player A takes 3 matches (remains 10)
- Player B takes 2 matches (remains 8)
- Player A takes 3 matches (remains 5)
- Player B takes 1 match (remains 4)
- Player A takes 2 matches (remains 2)
- Player B takes 2 matches (remains 0, WIN)

Problem Specification

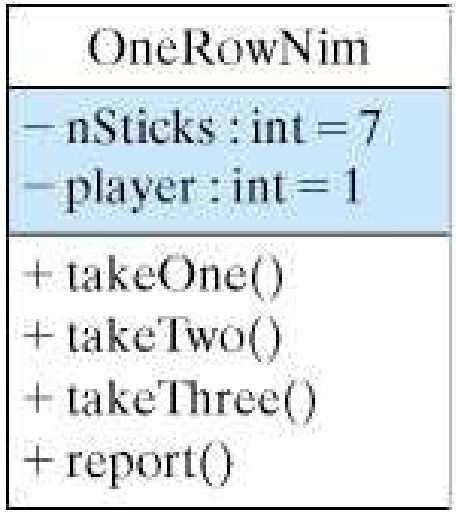
- Design a program that simulates the game of One-Row Nim with a row of sticks.
- A OneRowNim object will keep track of how many sticks remain and whose turn it is.
- A OneRowNim object should allow a player to pick up 1, 2, or 3 sticks.
- A OneRowNim object should know when the game is over and who won the game.

Problem Decomposition

- What *objects* do we need?
- The **OneRowNim** object will represent and manage the game.
- We design **OneRowNim** to be used with different kinds of interfaces.

Class Design: OneRowNim

- State:
  - Two int variables, nSticks and player
  - nSticks keeps tracks of the remaining sticks.
  - player keeps track of whose turn it is.
- Methods:
  - A takeOne() method to pick up 1 stick.
  - A takeTwo() method to pick up 2 sticks.
  - A takeThree() method to pick up 3 sticks.
  - A report() method describes the game's state.



OneRowNim Class Specification

- Class Name: OneRowNim
  - Role: To represent and simulate a One-Row Nim game
- Information Needed (object attributes)
  - **nSticks**: Stores the number of sticks left (private)
  - **player** : Stores whose turn it is (private)
- Manipulations Needed (public methods)
  - takeOne(), takeTwo(), takeThree()-- Methods to pick up 1, 2, or 3 sticks.
  - report(): A method to report the current state of the game (nSticks and player)

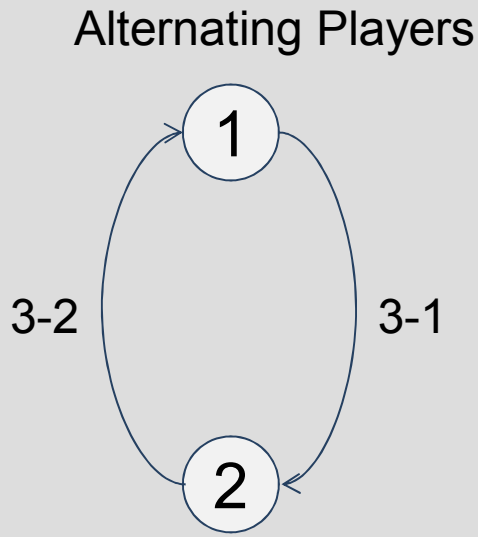
OneRowNim **Class Definition (1/2)**

```
public class OneRowNim
{ private int nSticks = 7;  // Start with 7 sticks.
  private int player = 1;   // Player 1 plays first.

  public void takeOne()
  { nSticks = nSticks - 1;
    player = 3 - player;
  } // takeOne()

  public void takeTwo()
  { nSticks = nSticks - 2;
    player = 3 - player;
  } // takeTwo()

  public void takeThree()
  { nSticks = nSticks - 3;
    player = 3 - player;
  }  // takeThree()
```



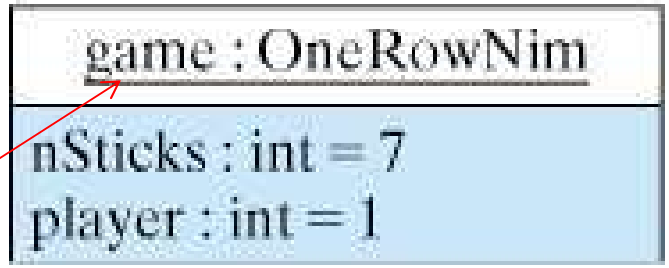
OneRowNim **Class Definition (2/2)**

```
  public void report()
  { System.out.println(
      "Number of sticks left: " + nSticks);
    System.out.println(
      "Next turn by player " + player);
  } // report()
} // OneRowNim1 class
```

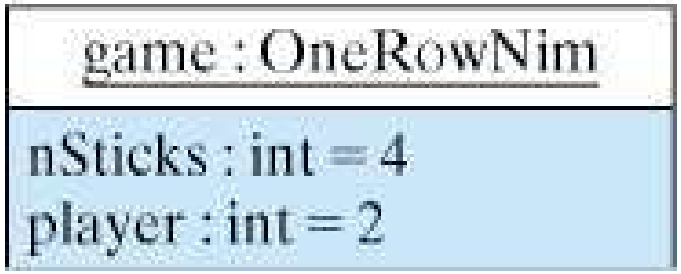
The OneRowNim Class

- A class is a blueprint. In this case every OneRowNim created will…
  - Have 7 sticks.
  - Player 1 will have the first turn.

Object Diagram
표기법 : objectName:className



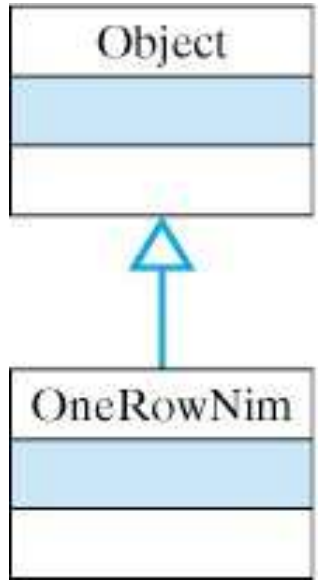
- But after calling the game.takeThree() method, the game object will change to:



The Class Header

- Example:

```
public class OneRowNim
```



- In General:

ClassModifiers$_{opt}$ class *ClassName Pedigree*$_{opt}$

```
public class OneRowNim extends Object
```

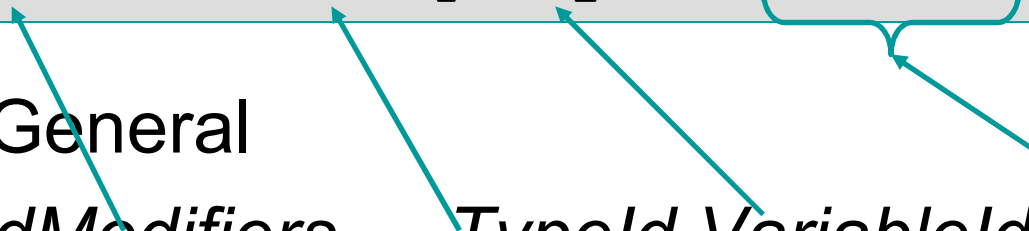
Identifiers

- An *identifier* is a name for a variable, method, or class.
- **Rule**: An identifier in Java must begin with a letter, and may consist of any number of letters, digits, and underscore (_) characters.
- Legal: OneRowNim, takeOne, nSticks
- Illegal: One Row Nim, 5sticks, game$, n!

Declaring Instance Variables

- Examples:

```
                    // Instance variables
private int nSticks = 7;
private int player = 1;
```



- In General

  FieldModifiers$_{opt}$ *TypeId VariableId Initializer*$_{opt}$

- Fields or instance variables have *class scope*. Their names can be used anywhere within the class.

Public/Private Access

- Instance variables should usually be declared private. This makes them inaccessible to other objects.
- Generally, public methods are used to provide carefully controlled access to the private variables.
- An object's public methods make up its *interface* -- that part of its makeup that is accessible to other objects.

Public/Private Access (cont)

- Possible Error: Public instance variables can lead to an inconsistent state
- Example: Suppose we make nSticks and player public variables.

```
nim.nSticks = -1;   // Inconsistent
nim.player = 0;     // State
```

- The proper way to change the game's state:

```
nim.takeOne();  // takeOne() is public method
```

Java's Accessibility Rules

- Packages contain classes which contain members (methods and fields).
- Access is determined from the top down.
- If no explicit declaration given, a default is used.

Type of Entity	Declaration	Accessibility Rule
Package	N/A	Accessibility determined by the system.
Class	public	Accessible if its package is accessible.
	default	Accessible only within its package.
Member (field or method) of an accessible class	public	Accessible to all other objects.
	protected	Accessible to its subclasses and to other classes in its package.
	private	Accessible only within the class.
	default	Accessible only within the package.

Initializer Expressions

- General Form: *Variable = expression*
- The *expression* on the right of the *assignment operator (=)* is evaluated and its value is stored in the *variable* on the left.
- Examples:

```
private int nSticks = 7;
private int player = "joe";    // Type error
```

- Type error: You can't assign a String value to an int variable

Method Definition

- Example

```
public void MethodName()  // Method Header
{                         // Start of method body
}                         // End of method body
```

- The Method Header

MethodModifiers$_{opt}$ *ResultType MethodName* (*ParameterList*)

public static	void	main	(String argv[])
public	void	takeOne	()
public	void	takeTwo	()
public	void	report	()

Method Definition

```
public void takeOne()
{
    nSticks = nSticks - 1;
    player = 3 - player;
} // takeOne()
```

Header: This method, named *takeOne*, is accessible to other objects *(public)*, and does not return a value *(void)*.

Body: a block of statements that removes one stick and changes the player's turn

Designing Methods

- The public methods serve as a class's interface.
- If a method is intended to be used to communicate with or pass information to an object, it should be declared public.
- A class's methods have *class scope*. They can be used anywhere within the class.
- Methods that do not return a value should be declared void.

The Simple Assignment Statement

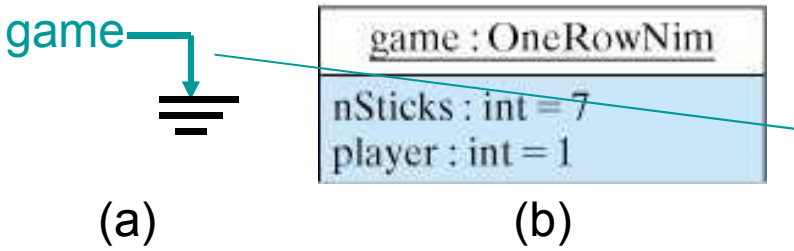
- General Form: *VariableName = Expression*
- The *expression* on the right of the *assignment operator (=)* is evaluated and its value is stored in the *variable* on the left.
- Examples:

```
nSticks = nSticks - 1;
player = 3 - player;
player = "joe";    // Type error
```

  - Type error: The value being assigned must be the same type as the variable.

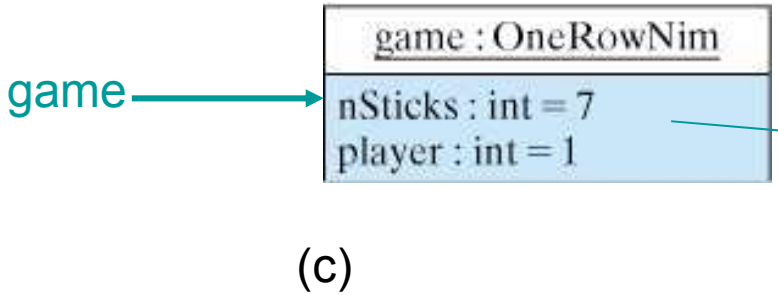
Creating OneRowNim **Instances**

- A *reference variable* refers to an object by storing the address of the object.




```
// Declare a reference variable
OneRowNim game;
```

(a) The reference variable, game, will refer to a OneRowNim objet, (b),
but its initial value is null.




```
// Create an instance
game = new OneRowNim();
```

(c) After instantiation, game refers to a OneRowNim object

Using OneRowNim **Objects**

- Objects are used by calling one of their public methods:

```
game.report();  // Tell game to report
game.takeOne(); // Tell game to take one stick away
```

OneRowNimTester Application

- The OneRowNimTester is an application with main() method.

```
public class OneRowNimTester
{
  public static void main(String args[])
  {   OneRowNim1 game = new OneRowNim();
      game.report();
      game.takeThree();
      game.report();
      game.takeThree();
      game.report();
      game.takeOne();
      game.report();
  } //main()
}
```

[LAB2] 예외처리 실습

- Power JAVA 2판, 21장 LAB 2번, p526
- 사용자가 입력한 값들을 크기가 10인 배열에 저장하여서 이 값들의 평균값을 계산하는 프로그램을 작성하여보자. 실행과정에서 발생할 수 있는 다음과 같은 예외들을 모두 처리하도록 프로그램하여라.
  - ArithmeticException
  - NegativeArraySizeException
  - ArrayIndexOutOfBoundException

예외처리 전 코드

```
import java.util.Scanner;

public class ExceptionTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int[] list;
        int sum=0, count;

        Scanner sc = new Scanner(System.in);
        System.out.print("정수의 개수 : ");
        count = sc.nextInt();
        list = new int[count];

        for (int i=0 ; i < count ; i++) {
            System.out.print("정수를 입력하시오 : ");
            list[i] = sc.nextInt();
        }

        for (int i=0 ; i < count ; i++) {
            sum += list[i];
        }
        System.out.println("평균은 " + sum / count);

    }
}
```

예외처리 후 코드

Homework#2

은행 계좌를 나타내는 BankAccount 라는 이름의 클래스를 설계하여 UML클래스 다이어그램을 그리고 JAVA로 구현하라.

- BankAccount 클래스는 이름, 계좌번호, 잔액, 이자율을 나타내는 필드를 가진다.
- BankAccount 클래스의 생성자를 overloading하라. 생성자는 모든 데이터를 받을 수도 있고, 아니면 하나도 받지 않을 수 있다.
- BankAccount 클래스는 입금을 나타내는 deposit() 메소드와 출금을 나타내는 withdraw() 메소드를 가진다.
- withdraw()에서 인출 금액이 잔액보다 크면 NegativeBalanceException을 발생시킨다. BankAccount 클래스를 테스트하는 BankAccountTest 클래스를 작성하고 발생되는 예외를 try/catch 문을 이용하여 처리하여 보자

과제제출 및 기한

- 과제 제출방법
  - Eclipse에서 hw2를 위한 프로젝트를 만들어 지시에 따라 프로그램을 작성
  - 프로젝트를 내보내기(Export) 기능을 사용하여 폴더에 저장
  - 프로젝트 폴더와 UML 다이어그램을 포함하여 OODhw02_학번 이름으로 압축하여 이예나 조교 <yenabanana@gmail.com> 메일로 보낼 것. 메일 보낼 때 자신의 학번과 이름을 반드시 내용에 포함해서 보낼 것
    - 학번은 자신의 학번을 의미
    - 예: 학번이 201023456인 학생은 프로젝트명이 OODhw02_201023456
- 제출기한
  - 다음 실습일 전날 밤 12시까지