실습 12

예외 처리 및 입출력 스트림

목 적

- 예외(exception)과 에러 처리에 대한 이해
- 예외 전파(exception propagation)에 대한 이해
- 입출력 스트림의 타입에 대한 이해

유의 사항

- Eclipse 를 사용하여 "**학번-Lab12**" 이름으로 프로젝트를 생성하여 실습한다.
- 실습이 끝나면 모든 프로그램의 최종본을 제출하기 위해 파일 시스템에서 "**학번-Lab12**" 이름의 Eclipse 프로젝트 폴더를 찾은 후, 프로젝트 폴더 전체를 압축(zip)하여 GitHub 저장소에 저장하고 GitHub 저장소의 URL 을 제출한다.

실습결과 응답 링크: https://forms.gle/Uk79VSFHpzvsFiHi8

에러가 아닌 예외(exception)

CountLetters.java는 단어를 읽어서 각 알파벳 문자가 단어 속에 나타난 회수를 출력하는 프로그램이다(알파벳 문자의 대소문자를 구분하지 않음). 프로그램을 디렉토리에 저장하고 이해한 후 컴파일 및 실행한다. 프로그램에서 단어를 먼저 모두 대문자로 변환하고, 각 문자로부터 'A'를 빼서 0과 25 사이의 수로 변환한 후 그 수를 인덱스로 사용한 것에 주목하라. 프로그램은 각 문자가 실제 문자인지를 판단하는 시험이 결여되어 있다.

1. CountLetters를 실행하고, 빈칸과 구두점 그리고 여러 개의 단어로 된 구문을 입력한다. 이 경우 알파벳문자(a-z와 A-Z) 이외의 문자는 0과 25 사이의 범위를 벗어난 인덱스 값으로 변환하게 되어 ArrayIndexOutOfBoundsException을 발생(throw) 시킨다. 이러한 알파벳 이외의 문자는 입력을 허용하되 문자 수를 셀 때 무시하는 것이 더 바람직할 것이다. 문자를 대문자로 변환한 뒤 그 문자가 'A'와 'Z' 사이에 있는지 시험하여 알파벳 이외의 문자를 무시할 수 있지만 ArrayIndexOutOfBoundsException을 catch하여 아무 것도 실행하지 않는 방법으로 무시할 수도 있다. 여기서는 후자의 방법을 택해 다음과 같이 프로그램을 수정한다.

   - 첫번째 for 루프를 try 문으로 옮긴다.
   - 예외를 캐치하지만 아무 것도 하지 않는 catch문을 추가한다.

   프로그램을 컴파일하고 실행한다.

2. "Not a letter"와 같은 유용한 메시지와 발생한 예외를 출력하도록 catch문 블록을 수정한다. 디버깅을 위해서는 발생한 예외를 출력하는 것이 좋지만, 에러로 간주하지 않는 예외인 경우 보통 예외를 출력하고 싶지 않을 것이다. 예외를 출력하는 대신 예외를 야기한 문자를 출력하도록 프린트문을 수정한다. 프로그램을 다시 실행한다. 출력이 훨씬 더 좋게 보일 것이다.

3. 모든 프로그램의 최종본이 맞는지 확인한다.

```
// ****************************************************************
// CountLetters.java
//
// Reads a word from the standard input and prints the number of
// occurrences of each letter in that word.
//
// ****************************************************************

import java.util.Scanner;

public class CountLetters
{
    public static void main(String[] args)
    {
      int[] counts = new int[26];

      Scanner in = new Scanner(System.in);

      //get word from user
      System.out.print("Enter a single word (letters only, please): ");
      String word = in.next();

      //convert to all upper case
      word = word.toUpperCase();

      //count frequency of each letter in string
      for (int i=0; i < word.length(); i++)
          counts[word.charAt(i)-'A']++;

      //print frequencies
      System.out.println();
      for (int i=0; i < counts.length; i++)
          if (counts [i] != 0)
            System.out.println((char)(i +'A') + ": " + counts[i]);

    }

}
```

예외 쓰로잉

Factorials.java는 MathUtils 클래스의 factorial 메쏘드를 이용하여 사용자가 입력한 정수의 팩토리얼을 계산하는 프로그램이다. 두 프로그램을 디렉토리에 저장하고 살펴본 뒤 컴파일하고 실행하여 작동을 이해한다. 프로그램을 여러 개의 양의 정수 값으로 시험해 본 후, 음의 정수 값으로 시험해 본다. 프로그램은 작은 양수(< 17)에 대해서는 정상적으로 작동하지만 큰 양수가 주어지면 큰 음수를 반환하며 음수가 주어지면 항상 1을 반환한다.

1. 팩토리얼은 수학적으로 음수에 대하여 정의되어 있지 않기 때문에 프로그램이 음수의 팩토리얼 값으로 1을 반환하는 것은 옳지 않다. 문제 해결을 위해 매개변수의 음수여부를 검사하도록 factorial 메쏘드를 수정할 수 있지만, factorial 메쏘드는 항상 값을 반환하기 때문에 에러 메시지를 출력한다 하더라도 값을 반환하는 것 자체가 잘못된 것이다. factorial 메쏘드는 값을 반환하는 대신 무언가 잘못되어 계산을 완료할 수 없음을 나타내는 예외를 쓰로우 해야한다. 여러 분 자신의 예외 클래스를 정의하여 사용할 수도 있지만 이 경우에 적합한 IllegalArgumentException(RuntimeException 상속) 예외가 이미 존재한다. 다음 지시에 따라 프로그램을 수정한다.

    - factorial 메쏘드가 IllegalArgumentException을 쓰로우 할 수 있다는 것을 알려 주도록 factorial 메쏘드의 헤더를 수정한다.
    - 매개변수의 값을 검사해서 음수이면 IllegalArgumentException을 쓰로우하도록 factorial 메쏘드를 수정한다. IllegalArgumentException 예외를 쓰로우할 때는 IllegalArgumentException 클래스의 인스턴스를 쓰로우해야 한다는 것에 주의한다. IllegalArgumentException 생성자는 문자열 매개변수를 요구하는데, 생성자의 매개변수로 문제가 무엇인지 나타낸 문자열을 주도록 한다.
    - 수정한 프로그램을 컴파일하고 실행한다. 음수를 입력하면 예외가 쓰로우되어 프로그램이 종료된다. 그 이유는 프로그램에서 예외를 캐치하지 않았기 때문에 main 메쏘드에서 예외를 쓰로우하게 되고 이것은 런타임 에러를 야기하게 되어 프로그램이 종료되는 것이다.
    - factorial 메쏘드가 쓰로우한 예외를 캐치해서 출력한 후 루프를 다시 계속 실행하도록 Factorials 클래스의 main 메쏘드를 수정한다. try와 catch문을 어디에 어떻게 넣어야 하는지 주의깊게 생각하라.

2. 17 이상인 값에 대하여 음수를 반환하는 것도 잘못된 것이다. 16보다 큰 수의 팩토리얼은 정수 타입으로 나타낼 수 없는 매우 큰 수가 나와 오버플로우가 발생하기 때문이다. 이 경우도 IllegalArgumentException으로 생각할 수 있다. 따라서 factorial 메쏘드가 음수이거나 17 이상의 수인 경우 예외 생성자에게 문제 원인에 대한 적절한 메시지를 매개변수로 주어 IllegalArgumentException을 쓰로우하도록 factorial 메쏘드를 수정한다.

3. 모든 프로그램의 최종본이 맞는지 확인한다.

```
// ********************************************************************
// Factorials.java
//
// Reads integers from the user and prints the factorial of each.
//
// ********************************************************************

import java.util.Scanner;

public class Factorials
{
    public static void main(String[] args)
    {
      char keepGoing = 'y';

      while (keepGoing == 'y' || keepGoing == 'Y')
          {
            System.out.print("Enter an integer: ");
            int val = in.nextInt();
            System.out.println("Factorial(" + val + ") = "
                           + MathUtils.factorial(val));
            System.out.print("Another factorial? (y/n) ");
            keepGoing = in.next().charAt(0);
          }
    }
}



// ********************************************************************
// MathUtils.java
//
// Provides static mathematical utility functions.
//
// ********************************************************************

public class MathUtils
{
    //-----------------------------------------------------------------
    // Returns the factorial of the argument given
    //-----------------------------------------------------------------
    public static int factorial(int n)
    {
      int fac = 1;
      for (int i=n; i>0; i--)
          fac *= i;
      return fac;
    }
}
```

BufferedReader 클래스의 사용

Java 2 플랫폼의 API 규격에서 BufferedReader 클래스에 대한 정의를 살펴본다. BufferedReader 클래스의 가장 유용한 메쏘드는 한 줄을 읽어 문자열로 반환하는 readLine 메쏘드이다. 설명한 것처럼 BufferedReader 클래스는 다음과 같이 입력 스트림을 이용하여 만들어진다

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
```

1. 여러 줄을 읽어서 다시 출력하는 프로그램을 BufferedReader 클래스를 사용하여 작성하라. 프로그램은 사용자가 "quit"을 입력하면 실행을 종료하고, IOException이 발생하면 단순히 예외를 출력한다. 문자열 객체를 비교할 때 equals 메쏘드를 사용해야 하는 것에 주의하라.

2. StringTokenizer 클래스를 사용하여 입력된 줄의 문자열에 있는 단어의 수를 알아내고 한 줄에 하나씩 각 단어를 출력한다. 단어는 빈 칸으로 구분하며, Java API 설명을 참고하여 StringTokenizer 클래스에 어떤 메쏘드가 있는지 살펴 본다. 입력된 각 줄의 문자열을 여러 부분으로 나눌 때 보통 StringTokenizer 클래스가 유용하다. 입력된 각 줄을 다음과 같이 처리한다.

   - 객체를 생성하고 입력된 문자열로 초기화 한다.
   - StringTokenizer 클래스의 countTokens 메쏘드를 사용하여 각 줄에 몇 개의 단어가 있는지 알아내고 이 수를 출력한다(예, "There are 3 words in this line.")
   - 루프에서 토큰이 없을 때까지 하나씩 출력한다.

3. 모든 프로그램의 최종본이 맞는지 확인한다.

텍스트 파일에 출력하기 (재고목록 수정)

UpdateInventory.java는 inventory.dat 파일로부터 재고 물품 목록을 읽고 원소가 InventoryItem 타입인 배열에 저장한 후 이 배열의 내용을 출력하는 프로그램이다.

프로그램을 컴파일하고 실행하여 작동을 이해한다.

물품을 현재의 재고 목록에 추가하고 추가된 물품에 대한 새로운 재고 데이터를 inventory.dat 파일에 다시 저장할 수 있도록 프로그램을 수정한다. 현재의 재고 목록에 물품을 추가하기 위해, 프로그램은 각 물품을 얼마나 추가할 것인지 표준입력장치로부터 읽어서 그 물품의 현재 재고(InventoryItem 클래스의 units 변수)에 더한다. 재고추가를 위해 InventoryItem 클래스에 다음 4개의 메쏘드를 정의한다

- restock 메쏘드: 재고에 추가될 물품의 개수를 매개변수로 갖는다. 이 수는 units 변수에 더해진다.
- 3개의 호출(accessor) 메쏘드getName, getUnits, getPrice: 각각 재고 물품의 이름, 재고 개수, 가격을 반환

UpdateInventory.java를 다음과 같이 수정한다.

1. for 루프의 println 문 앞에서
   a. getName 메쏘드로 얻은 물품 이름을 프롬프트에 주고 추가할 수량을 묻는다.
   b. 수를 읽어 들인다. (InputStreamReader 클래스와 BufferedReader 클래스를 사용할 것)
   c. restock 메쏘드를 사용하여 재고 물품 개수를 갱신한다.

2. 지금까지 수정한 프로그램을 컴파일하고 실행해 본다.

3. 다음과 같이 갱신된 재고 정보를 파일에 다시 기록하는 코드를 추가한다.
   a. 루프가 시작하기 전에 출력 파일 스트림을 만든다. (출력 파일 스트림 생성은 TestData.java 소스파일을 참고하라.)
   b. 원래의 물품재고 파일 포맷과 같은 포맷으로 루프에서 물품 재고 정보를 기록한다. (accessor 메쏘드를 이용하라.)
   c. 루프 종료 후 출력 파일 스트림을 닫는다.

4. 프로그램을 컴파일하고 실행한다. 파일이 갱신되었는지 inventory.dat의 내용을 살펴본다.

5. 모든 프로그램의 최종본이 맞는지 확인한다.

```
//********************************************************************************
**
//   UpdateInventory.java
//
//   Demonstrates the use of character file I/O.
//
********************************************************************************
*

import java.io.*;
import java.util.StringTokenizer;

public class UpdateInventory
{
    // -----------------------------------------------------------------
    //  Reads data about a store inventory from an input file, creating
    //  an array of InventoryItem objects. For each inventory item, gets
    //  the number of additional units and updates the total units
    //  available (restocks), then writes the updated inventory back to
    //  the file.
    // -----------------------------------------------------------------
    public static void main (String[] args)
    {
      final int MAX = 100;
      InventoryItem[] items = new InventoryItem[MAX];
      StringTokenizer tokenizer;
      String line, name, file = "inventory.dat";
      int units, count = 0;
      int addedUnits;
      float price;

      try
      {
        FileReader fr = new FileReader (file);
        BufferedReader inFile = new BufferedReader (fr);

        line = inFile.readLine();
        while (line != null)
        {
          tokenizer = new StringTokenizer (line);
          name = tokenizer.nextToken();
          try
          {
            units = Integer.parseInt (tokenizer.nextToken());
            price = Float.parseFloat (tokenizer.nextToken());
            items[count++] = new InventoryItem (name, units, price);
          }
          catch (NumberFormatException exception)
          {
            System.out.println ("Error in input. Line ignored.");
            System.out.println (line);
          }
          line = inFile.readLine();
        }
        inFile.close();

        System.out.println("\nEnter the number of additional units of each item: ");

        // Set up the output file stream

        for (int scan = 0; scan < count; scan++)
        {
          System.out.println (items[scan]);
        }
```

```
        // Close the output file stream

      }
      catch (FileNotFoundException exception)
      {
        System.out.println ("The file " + file + " was not found.");
      }
      catch (IOException exception)
      {
        System.out.println (exception);
      }
    }
}
```

```
// ********************************************************************
//   InventoryItem.java
//
//   Represents an item in the inventory.
// ********************************************************************

import java.text.DecimalFormat;

public class InventoryItem
{
    private String name;
    private int units;     // number of available units of this item
    private float price;   // price per unit of this item
    private DecimalFormat fmt;

    // -----------------------------------------------------------------
    //  Sets up this item with the specified information.
    // -----------------------------------------------------------------
    public InventoryItem (String itemName, int numUnits, float cost)
    {
      name = itemName;
      units = numUnits;
      price = cost;
      fmt = new DecimalFormat ("0.##");
    }

    // -----------------------------------------------------------------
    //  Returns information about this item as a string
    // -----------------------------------------------------------------
    public String toString()
    {
      return name + ":\t" + units + " at " + price + " = " +
          fmt.format ((units * price));
    }

    // -----------------------------------------------------------------
    //  Returns the name of the item.
    // -----------------------------------------------------------------
    public String getName()
    {
      return name;
    }
```

```
    // ----------------------------------------------------------------
    //   Returns the number of available units of the item
    // ----------------------------------------------------------------
    public int getUnits ()
    {
      return units;
    }

    // ----------------------------------------------------------------
    //   Returns the price of the item.
    // ----------------------------------------------------------------
    public float getPrice()
    {
      return price;
    }

    // ----------------------------------------------------------------
    //   Adds the specified number to the available units
    // ----------------------------------------------------------------
    public void restock (int addedUnits)
    {
      units += addedUnits;
    }
}
```

inventory.dat

```
Widget 14 3.35
Spoke 132 0.32
Wrap 58 1.92
Thing 28 4.17
Brace 25 1.75
Clip 409 0.12
Cog 142 2.08
```