|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **[유헬스2\_11주\_실습과제]** | **학번:** | **20175105** | **이름:** | **곽영주** |

* 예외처리 : try, catch, finally

1. 예외가 발생한 곳에서 처리

import java.util.\*;

public class DivideByZero {

public static void main(String[] args){

int x,y;

Scanner sc=new Scanner(System.*in*);

System.*out*.print("피젯수: "); x=sc.nextInt();

System.*out*.print("젯수: "); y=sc.nextInt();

try{ //예외 발생 가능한 문장을 try 블록에 작성

int result=x/y; //예외 발생

System.*out*.println(" x/y = " + result);

}

catch(ArithmeticException e){ //발생한 예외를 처리

System.*out*.println("0으로 나눌 수 없습니다");

}

System.*out*.println("프로그램을 종료합니다.");

}

}

// 프로그램 실행 시 y값을 0과 0이 아닌 값으로 각각 입력하였을 때 결과를 제시하시오

|  |
| --- |
| **[실행결과]** |

2. 예외를 상위 메소드로 전달

public class Account {

private int balance; //잔액

public void setBalance(int balance) {

this.balance = balance;

}

//발생한 예외를 호출한 곳으로 전달, 현재 메소드에서 발생하는 예외종류를 throws 절로 표시

int withdraw(int amount) throws Exception {

if (balance < amount)

//throw절을 이용하여 인위적으로 예외객체 생성

throw new Exception("잔액이 부족합니다.");

balance -= amount;

return amount;

}

}

public class ExceptionExample {

public static void main(String[] args) {

Account obj = new Account();

obj.setBalance(100000);

try { //예외 발생 가능성이 있는 문장을 try블럭에 작성

int amount = obj.withdraw(500000000); //예외 발생 메소드

System.*out*.println("인출액:" + amount);

}

catch (Exception e) { //발생된 예외 처리

System.*out*.println(e.getMessage()); //오류 메시지 출력

}

finally{ //예외 발생과 상관없이 항상 실행

System.*out*.println("실행을 종료합니다.");

}

}

}

// withdraw()메소드 실매개 변수 값을 5000으로 수정하기 전과 후의 결과를 제시하시오

|  |
| --- |
| **[실행결과]** |

* 사용자 정의 예외 클래스
* Exception 을 상속받아 구현
* 생성자에서 super(msg)로 구현
* msg에 저장된 문자열은 메소드 getMessage()에 의해 반환
  + 생성된 예외의 발생
* 생성된 예외 클래스는 필요한 경우 new를 사용하여 예외 발생(throws)
* 메소드 선언에서 throws 사용
* 발생되는 예외가 여러 개라면 여러 예외 유형을 쉼표로 구분하여 기술

class ZeroException extends Exception{ //사용자 정의 예외 클래스

public ZeroException(){

super("0으로 나눌 수는 없슴");

}}

public class {

public static void main(String[] args) {

double result;

try{

result= quotient(20, 0);}

catch(ZeroException e){ //사용자 정의 예외 처리

System.out.println(e. getMessage());

}

}

public static double quotient(int n, int d) throws ZeroException{

if(d==0)

throw new ZeroException(); //예외 발생

return (double)n/d;}

}

|  |
| --- |
| **[실행결과]** |

* 제네릭 프로그래밍
* 다양한 타입의 객체를 동일한 코드로 처리하는 기법
* 제네릭은 컬렉션 라이브러리에 많이 사용
* 제네릭은 **클래스와 인터페이스에만 적용, 자바 기본 타입을 실제 타입으로 사용할 수 없슴**

1. 제네릭 클래스

class Box<T> { //T는 데이터 타입을 의미, < >안에 표기, generic

private T data; //T 타입을 자료형으로 갖는 data 필드

public Box(T data){ //T 타입을 형식매개변수로 사용

this.data = data;

}

public void set(T data) {this.data = data;}

public T get() {return data;}

}

public class BoxTest {

public static void main(String[] args){

// 다이아 몬드 생성자 호출 시 타입 인수 생략 가능 SE7 버전부터 적용

Box<String> sg=new Box<>("Generic"); // 문자열을 저장하는 Box 클래스의 객체 생성

Box<Integer> ig=new Box<Integer>(340); //정수를 저장하는 Box 클래스의 객체

System.*out.println("<String>" + sg.get());*

System.*out.println("<Integer>" + ig.get());* }

}

|  |
| --- |
| **[실행결과]** |

1. 제네릭 메소드

**public class** MyArrayAlg {

**public** **static** <T> T getLast(T[] a) {

**return** a[a.length - 1];

}

}

**public** **class** MyArrayAlgTest {

**public** **static** **void** main(String[] args) {

String[] language = { "C++", "C#", "JAVA" };

String last = MyArrayAlg.*getLast*(language); // last는 “JAVA"

System.***out***.println(last);

}

}

|  |
| --- |
| **[실행결과]** |

1. 한정된 타입 매개변수

public class MyArrayAlg{

public static <T extends Comparable> T getMax(T[] a){

if (a == null || a.length == 0) return null;

T largest = a[0];

for (int i = 1; i < a.length; i++)

if (largest.compareTo(a[i]) < 0)

largest = a[i];

return largest;

}}

public class MyArrayAlgTest {

public static void main(String[] args) {

Integer[] language = { 11, 58, 26, 89, 64, 120,890};

System.*out.println(MyArrayAlg.getMax(language));*

*// double 형 무명배열을 매개변수로 전달한 후 결과 확인*

}}

|  |
| --- |
| [실행결과] |
| [double형 실행결과]  **public** **class** MyArrayAlgTes {  **public** **static** **void** main(String[] args) {  Integer[] language = {11, 58, 26, 89, 64, 120, 890};  System.***out***.println(MyArrayAlg1.*getMax*(**new** Double[] {2.3, 4.5, 6.3, 12.3, 56.3}));  }  } |

* 내용점검

1. ArrayIndexOutOfBoundsException 예외를 발생시키는 프로그램을 작성하고 try/catch 블록을 이용하여 처리하여 보라.

Int[] dim = new int[10];

**try** {

dim[10] = 30;

}

**catch** (ArrayIndexOutOfBoundsException e) {

System.out.println(e.getMessage()); // 콘솔에 오류 메시지를 출력한다.

}

1. 다음과 같은 예외 처리기로 잡을 수 있는 예외는 어떤 종류인가? 그리고 이런 종류의 예외 처리기가 바람직하지 않은 이유는?

**catch** (Exception e) {

...

}

Exception의 하위클래스 모두 잡을 수 있으나 분간 할 수 없다.

1. 다음 코드에서 잘못된 부분을 지적하시오.

**try** {

...

} **catch** (ArithmeticException a) {

...

}

**catch** (Exception e) {

...

}

1. 다음 프로그램의 출력을 쓰시오.

**try** {

**int** n = Integer.*parseInt*("abc");

System.*out*.println("try");

}

**catch** (NumberFormatException e){

System.*out*.println("숫자 형식 오류");

}

**finally** {

System.*out*.println("finally");

}

숫자 형식 오류

finally

1. 다음 프로그램의 출력을 쓰시오.

**try** {

**int**[] array=**new** **int**[-10];

System.*out*.println("try");

}

**catch** (NumberFormatException e){

System.*out*.println("숫자 형식 오류");

}

**catch** (NegativeArraySizeException e){

System.*out*.println("배열 크기 음수 오류");

}

**catch** (Exception e){

System.*out*.println("오류");

}

배열 크기 음수 오류

1. 다음 프로그램의 출력을 쓰시오.

public class Test {

public static void throwit () {

System.out.print("A ");

throw new RuntimeException();

}

public static void main(String [] args) {

try {

System.out.print("B ");

throwit();

}

catch (Exception re ){

System.out.print("C ");

}

finally{

System.out.print("D ");

}

System.out.println("E ");

}

}

B A C D E

1. 다음 프로그램의 출력을 쓰시오.

public class Test{

public static void main(String [] args) {

try{

someMethod();

System.out.print("A");

}

catch (Exception ex){

System.out.print("B");

}

finally {

System.out.print("C");

}

System.out.print("D");

}

public static void someMethod() {}

}

ACD

1. 다음과 같은 문장에 의하여 발생하는 예외는 무엇인가?

(1) int[] anArray = new int[3];

System.out.println(anArray[3]);

예외) anArray[3] = null 값

(2) String[] strs = new String[3];

System.out.println(strs[0].length());

예외) strs[0] = null 값

(3) Integer.parseInt("abc");

예외) 문자열 안에 숫자가 아닌 것

(4) Object o = new Object();

Integer i = (Integer)o;

예외) Integer 값이 아닐경우

1. 다음 프로그램을 컴파일하여 보자.

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**try{**

sub();

**}**

**catch() {**

**}**

}

**public** **static** **void** sub() throws Exception {

**int**[] array = **new** **int**[10];

**try {**

**int** i = array[10];

**}**

Catch(){

}

}

}

(1) 위의 프로그램은 컴파일 시에 오류가 발생한다. 어떤 오류가 발생하는가?

(2) try/catch 블록을 사용하여서 예외를 처리하여 보라.

(3) throws 선언을 이용하여 예외를 처리하여 보라. Sub() Throws Exception

1. 다음은 Stack 클래스의 일부분이다.
2. Stack에 저장되는 데이터의 타입을 int 대신에 제네릭 타입으로 표시하여 보자.

**public** **class** <T>Stack{

**private** T[] stack;

**public** **void** push(T data) { ....}

**public** T pop() { ....}

}

1. String 타입의 데이터를 가지는 Stack을 생성하는 문장을 쓰시오.

stack<String> obj = new stack<>();

* 프로그램 과제

1. 음수가 전달되면 사용자 예외를 발생한다. 빈 공란을 채워 넣은 후 실행결과를 제시하시오

**//사용자 정의 예외 클래스 MyException**

class MyException {

public MyException(String str){

//형식매개변수 str을 수퍼클래스 생성자로 전달

}

}

public class MyExceptionTest {

**//사용자 정의 예외 MyException을 메소드를 호출한 곳으로 던진다**

public static void checkNegative(int no) {

if(no < 0)

**//사용자 정의 예외 MyException를 인위적으로 생성**

throw new MyException(no+ "는 사용 불가(음수데이터)");

else

System.*out*.println(no+ "는 사용 가능한 데이터");

}

public static void main(String[] args) {

{

*checkNegative*(10);

*checkNegative*(-1);

}catch(MyException e){

System.*out*.println(e.getMessage());

}

{

System.*out*.println("항상 실행되는 부분입니다.");

}

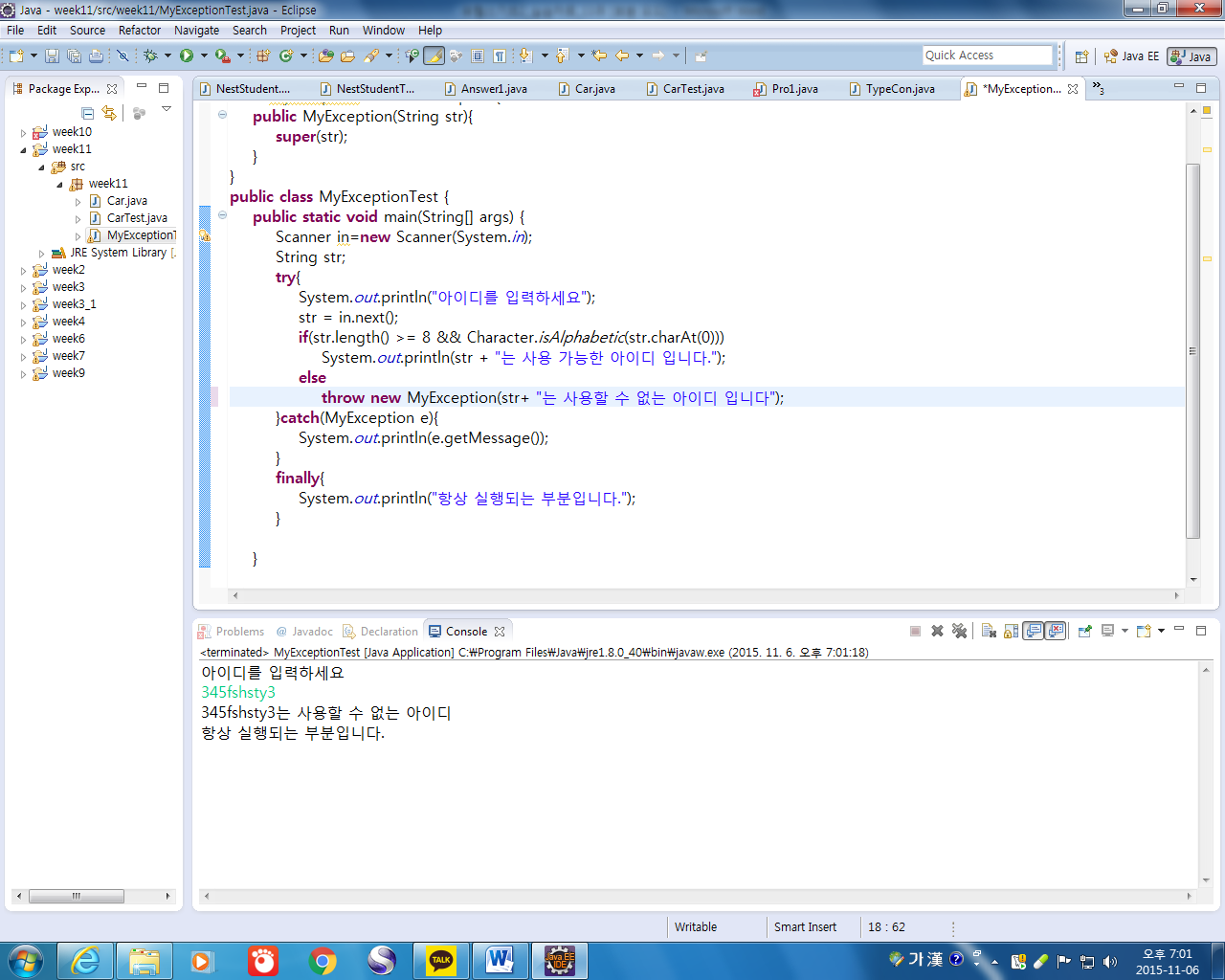
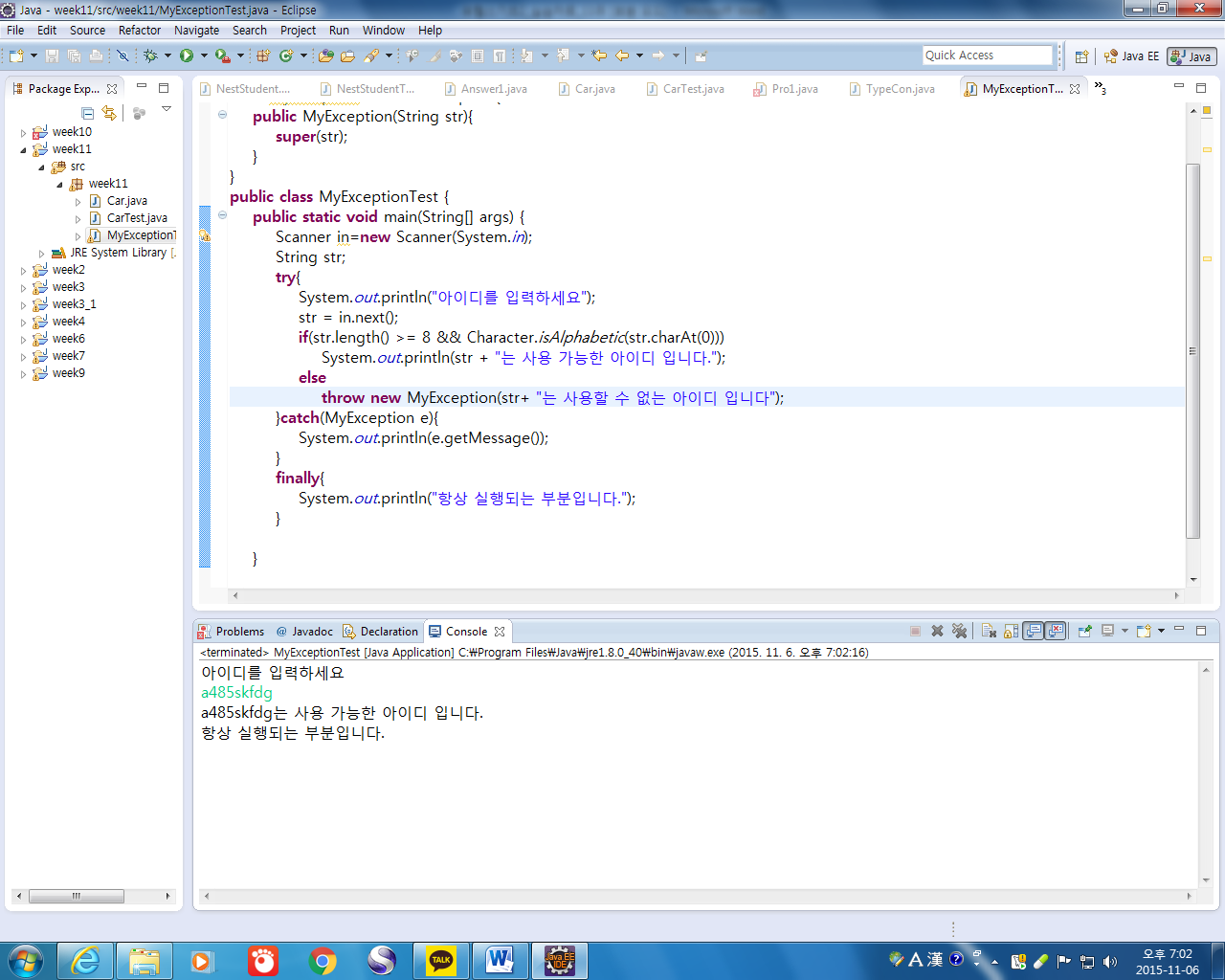
}

}

|  |
| --- |
| **[프로그램 소스]**  **public** **class** MyException **extends** Exception {  **public** MyException(String str) {  **super**(str);  }  }  **public** **class** MyExceptionTest {  **public** **static** **void** checkNegative(**int** no) **throws** MyException {  **if**(no < 0)  **throw** **new** MyException(no + "는 사용 불가(음수데이터)");  **else**  System.***out***.println(no + "는 사용 가능한 데이터");  }    **public** **static** **void** main(String[] args) {  **try** {  *checkNegative*(10);  *checkNegative*(-1);  }  **catch**(MyException e) {  System.***out***.println(e.getMessage());  }  **finally** {  System.***out***.println("항상 실행되는 부분입니다.");  }  }  } |
| **[실행결과]** |

1. 키보드로 입력 받은 문자열에 대하여 길이가 8미만이거나 첫 글자가 영문자가 아니면 NotUseId 사용자 예외를 발생시켜 “사용할 수 없는 아이디”를 출력하고, 길이가 8이상이고 첫 글자가 영문자 이면 “사용 가능한 아이디 입니다.” 를 출력한다.

[출력예시]

|  |
| --- |
| **[프로그램 소스]**  **public** **class** ID {  **public** **static** **void** NotUseld(String id) **throws** Exception {  **if**(8 > id.length() || 'a' > id.charAt(0) || id.charAt(0) > 'z') {  **throw** **new** Exception(id + "는 사용할 수 없는 아이디");  }  **else** {  System.***out***.println(id + "는 사용 가능한 아이디 입니다.");  }  }  }  **import** java.util.\*;  **public** **class** IDTest {  **public** **static** **void** main(String[] args) {  Scanner key = **new** Scanner(System.***in***);  ID obj = **new** ID();  System.***out***.println("아이디를 입력하세요");  **try** {  obj.*NotUseld*(key.next());  }  **catch**(Exception e) {  System.***out***.println(e.getMessage());  }  }  } |
| **[실행결과]** |

1. 타입 매개변수 T를 가지는 제네릭 클래스 MyArray를 제시된 조건대로 작성하고 테스트 하시오. Integer, Double, String 타입을 사용하도록 한다.

* 필드 : T 타입의 배열 선언
* 생성자 : 형식매개변수로 T타입의 배열을 받는다
* 형식매개변수 없으며 필드로 사용되는 배열에서 첫번째 원소를 반환하는 firstEle() 메소드
* 형식매개변수 없으며 필드로 사용되는 배열에서 마지막 원소를 반환하는 lastEle() 메소드
* 형식 매개변수와 반환값 없으며 필드로 사용되는 배열원소를 출력하는 display() 메소드

|  |
| --- |
| **[프로그램 소스]**  **public** **class** MyArray <T> {  **private** T[] dim;    **public** MyArray(T[] dim) {  **this**.dim = dim;  }    **public** T firsEle() {  **return** dim[0];  }    **public** T lastEle() {  **return** dim[dim.length-1];  }    **public** **void** display() {  **for**(**int** i=0; i<dim.length; i++) {  System.***out***.print(dim[i] + "\t");  }  }  }  **public** **class** MyArrayTest {  **public** **static** **void** main(String[] args) {    MyArray<Integer> obj1 = **new** MyArray<>(**new** Integer[] {1,2,3,4,5});  System.***out***.print("배열에서 첫 번째 원소 : ");  System.***out***.println(obj1.firsEle());  System.***out***.print("베열에서 마지막 원소 : ");  System.***out***.println(obj1.lastEle());  System.***out***.println("배열원소 출력");  obj1.display();  System.***out***.println();    MyArray<String> obj2 = **new** MyArray<>(**new** String[] {"Kwak","Yeong","Ju"});  System.***out***.print("\n배열에서 첫 번째 원소 : ");  System.***out***.println(obj2.firsEle());  System.***out***.print("베열에서 마지막 원소 : ");  System.***out***.println(obj2.lastEle());  System.***out***.println("배열원소 출력");  obj2.display();  System.***out***.println();    MyArray<Double> obj3 = **new** MyArray<>(**new** Double[] {1.2, 5.3, 12.5, 4.5});  System.***out***.print("\n배열에서 첫 번째 원소 : ");  System.***out***.println(obj3.firsEle());  System.***out***.print("베열에서 마지막 원소 : ");  System.***out***.println(obj3.lastEle());  System.***out***.println("배열원소 출력");  obj3.display();  System.***out***.println();  }  } |
| **[실행결과]** |

1. 클래스 안에서 하나의 메소드만 제네릭으로 만들어보자. 제네릭 메소드 a()를 가지는 클래스 Test를 정의하여 보자. 메소드 a()는 매개 변수의 클래스 이름을 출력한다. 객체 obj의 클래스 이름은 obj.getClass().getName()으로 출력할 수 있다. int나 float와 같은 기초형 값을 전달하여서 호출해보자. 어떤 값이 출력되는가?

|  |
| --- |
| **[프로그램 소스]**  **public** **class** Test {  **public** **static** <T> **void** a(T obj) {  System.***out***.println(obj.getClass().getName());  }    **public** **static** **void** main(String[] args) {  Integer in= **new** Integer(28);  Test.*a*(in);    Double dou = **new** Double(2.4);  Test.*a*(dou);    String str = **new** String("aa");  Test.*a*(str);  }  } |
| **[실행결과]** |