Object – Oriented Programming

Lab #09

Exception

- An *exception* represents an error condition that can occur during the normal course of program execution

- When an exception occurs, or is *thrown*, the normal sequence of flow is terminated. The exception-handling routine is then executed; we say thrown exception is *caught*

Introduction to Exception Handling

- **Java library software (or programmer-defined code) provides a mechanism that signals when something unusual happens**
  - This is called *throwing* an exception
- **In another place in the program, the programmer must provide code that deals with the exceptional case**
  - This is called *handling the exception*

Not Catching Exceptions

```
Scanner scanner = new Scanner(System.in);
System.out.println("Enter integer:");
int number = scanner.nextInt();
```

What would happen if the user enters a value such as the test 'ten' instead of 10?

Error message for invalid input

```
Exception in thread "main" java.lang.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:819)
        at java.util.Scanner.next(Scanner.java:1431)
        at java.util.Scanner.nextInt(Scanner.java:2040)
        at java.util.Scanner.nextInt(Scanner.java:2000)
        at Ch8Sample1.main(Ch8Sample1.java:35)
```

Not Catching Exceptions (Contd)

- **Two things occurred in this scenario**
  - Java threw an exception named InputMismatchException
  - Our program failed to catch the exception resulting in a crash

  - We can fix this by enclosing our code in a
    - ***try – catch*** block

try-throw-catch Basics

```
try {

    // some code to attempt
    //this code may throw an exception

} catch (Exception e){

    //catch the exception if it is thrown
    //do whatever you want with it.

}
```

Catching an Exception

try

catch

```
System.out.print(prompt);

try {

    age = scanner.nextInt( );

} catch (InputMismatchException e){

    System.out.println("Invalid Entry. "
                       +  "Please enter digits only");
}
```

try-catch Control Flow

Exception

```
try {
  <t-stmt-1>
  <t-stmt-2>
  <t-stmt-3>
  <t-stmt-4>
  . . .
  <t-stmt-n>

} catch (Exception e) {
  <c-stmt-1>
  . . .
  <c-stmt-m>
}
<next stmt>
```

Assume <t-stmt-3> throws an exception.

Remaining statements in the try block is skipped.

Statements in the catch block are executed.

And the execution continues to the next statement

No Exception

```
try {
  <t-stmt-1>
  <t-stmt-2>
  <t-stmt-3>
  <t-stmt-4>
  . . .
  <t-stmt-n>

} catch (Exception e) {
  <c-stmt-1>
  . . .
  <c-stmt-m>
}
<next stmt>
```

All statements in the try block are executed.

Statements in the catch block are skipped.

try-catch

```java
public class TryCatch {

    public static void main(String[] args){
        try{
            System.out.println(1);
            System.out.println(2);
            System.out.println(3);
            System.out.println(0/0);
            System.out.println(4);
        }
        catch(Exception e){
            System.out.println(e.getMessage());
        }
        System.out.println(6);
    }

}
```

```
1
2
3
/ by zero
6
```

try-catch (Contd)

- **getMessage() method**
  - Every exception has a String instance variable that contains some message, which typically identifies the reason for the exception
  - The getMessage() returns the detail message string

Exception Object

- **The two most important things about an exception object are its type (i.e., exception class) and the message it carries**
  - The message is sent along with the exception object as an instance variable
  - This message can be recovered with the accessor method *getMessage(),* so that the catch block can use the message

Exception Class

- **Numerous predefined exception classes are included in the standard packages that come with Java**
  - For example:
    - *IOException*
    - *NoSuchMethodException*
    - *FileNotFoundException*
  - Many exception classes must be imported in order to use them
    - *import java.io.IOException*

Exception Message type

- **An exception class can carry messages of any type**
- **An exception class constructor can be defined that takes an argument of another type**
  - It would stores its value in an instance variable
  - It would need to define accessor methods for this instance variable

Programmer-defined Exceptions

- Exception classes may be programmer-defined, but every such class must be a derived class of an already existing exception class
- The class *Exception* can be used as the base class, unless another exception class would be more suitable
- At least two constructors should be defined, sometimes more
- The exception class should allow for the fact that the method *getMessage()* is inherited

Programmer-defined Exceptions (Contd)

```
Public class MyException extends Exception{
    // variables

    public MyException(){
        super("default message");
        //perform other tasks
    }

    public MyException(VariableType var){
        super(var+ "rest of message");
        //perform other tasks
    }

    //other methods if needed
}
```

Preserve getMessage

- **For all predefined exception classes, *getMessage()* returns the string that is passed to its constructor as an argument**
  - Or it will return a default string if no argument is used with the constructor
- **This behavior must be preserved in all programmer-defined exception class**
  - A constructor must be included having a string parameter whose body begins with a call to *super*
  - The call to *super* must use the parameter as its argument
  - A no-argument constructor must also be included whose body begins with a call to *super*
  - This call to *super* must use a default string as its argument

Example

Display 9.5 **An Exception Class with an int Message**

```
public class BadNumberException extends Exception
{
    private int badNumber;

    public BadNumberException(int number)
    {
        super("BadNumberException");
        badNumber = number;
    }

    public BadNumberException()
    {
        super("BadNumberException");
    }

    public BadNumberException(String message)
    {
        super(message);
    }

    public int getBadNumber()
    {
        return badNumber;
    }
}
```

Multiple catch Blocks

- A single try-catch statement can include multiple catch blocks, one for each type of exception

```
try {
        …
        age = scanner.nextInt();
        …
        val = cal.get(id); //cal is a GregorianCalendar
        …
} catch (InputMismatchException e){
    …
} catch (ArrayIndexOutOfBoundsException e){
    …
}
```

Multiple catch Control Flow

Exception

```
try {
    <t-stmt-1>
    <t-stmt-2>
    <t-stmt-3>
    <t-stmt-4>
    . . .
    <t-stmt-n>
}
<catch-block-1>
<catch-block-2>
<catch-block-3>
. . .
<catch-block-m>
}
<next stmt>
```

Assume <t-stmt-3> throws an exception and <catch-block-3> is the matching block.

Remaining statements in the try block is skipped.

Statements in the matching catch block are executed.

No Exception

```
try {
    <t-stmt-1>
    <t-stmt-2>
    <t-stmt-3>
    <t-stmt-4>
    . . .
    <t-stmt-n>
}
<catch-block-1>
<catch-block-2>
<catch-block-3>
. . .
<catch-block-m>
}
<next stmt>
```

All statements in the try block are executed and throw no exceptions.

All catch blocks are skipped.

Important

- **When using multiple catch statements *always catch the more specific exceptions first***

  catch (Exception e)

  {…}

  catch (NegativeNumberException e)

  {…}

- **Because a *NegativeNumberException* is a type of *Exception,* all *NegativeNumberExceptions* will be caught by the first *catch* block before ever reaching the second block**
  - The catch block for *NegativeNumberException* will never be used!

- **For the correct ordering, simply reverse the two blocks**

The *finally* Block

- There are situations where we need to take certain actions regardless of whether an exception is thrown or not
- We place statements that must be executed regardless of exceptions in the *finally* block

try-catch-finally Control Flow

Exception

```
try {
  <t-stmt-1>
  . . .
  <t-stmt-i>
  . . .
  <t-stmt-n>
}
<catch-block-1>
. . .
<catch-block-i>
. . .
<catch-block-m>
} finally {
  . . .
}
<next stmt>
```

Assume <t-stmt-i> throws an exception and <catch-block-i> is the matching block.

finally block is executed.

No Exception

```
try {
  <t-stmt-1>
  . . .
  <t-stmt-i>
  . . .
  <t-stmt-n>
}
<catch-block-1>
. . .
<catch-block-i>
. . .
<catch-block-m>
} finally {
  . . .
}
<next stmt>
```

finally block is executed.

Self-Test (1)

- **Exception 클래스인 PowerFailureException 클래스를 정의할 것**
  - 음수값을 입력하면 발생하는 exception
  - 인자가 없는 생성자를 지녀야 함
  - 해당 생성자를 통해 exception이 발생할 경우 getMessage() 메소드는 "Power Failure"를 반환해야 함
  - 하나의 String type 인자를 가지는 생성자를 지녀야 함
  - 해당 생성자를 통해 exception이 발생할 경우 getMessage() 메소드는 생성자의 인자로 사용된 값을 반환함
- **Exception 클래스인 TooMuchStuffException 클래스를 정의할 것**
  - 10 이상의 숫자를 입력하면 발생하는 exception
  - 인자가 없는 생성자를 지녀야 함
  - 해당 생성자를 통해 exception이 발생할 경우 getMessage() 메소드는 "Too much stuff!"를 반환해야 함
  - 하나의 int type 인자를 가지는 생성자를 지녀야 함
  - 해당 생성자를 통해 exception이 발생할 경우 getNumber() 메소드는 생성자의 인자로 사용된 값을 반환함

Self-Test (1) (Contd)

Problems | @ Javadoc | Declaration | Console

SimpleException [Java Application] C:₩Program Files₩Java₩jre1.8.0_161₩bin₩javaw.exe (2018. 5. 2. 오후 8:23:

```
Enter the number 0 - 9
If you enter a negative number, PowerFailureException will occur
If you enter a positive number other than 0 - 9, TooMuchStuffException will occur
-200
Power Failure
End of try-catch statement

Enter the number 0 - 9
If you enter a negative number, PowerFailureException will occur
If you enter a positive number other than 0 - 9, TooMuchStuffException will occur
5
No exception has been occrued
End of try-catch statement

Enter the number 0 - 9
If you enter a negative number, PowerFailureException will occur
If you enter a positive number other than 0 - 9, TooMuchStuffException will occur
199
199 occurs TooMuchStuffException
End of try-catch statement

Enter the number 0 - 9
If you enter a negative number, PowerFailureException will occur
If you enter a positive number other than 0 - 9, TooMuchStuffException will occur
```

Propagating Exceptions

- **Instead of catching a thrown exception by using the try-catch statement, we can *propagate* the thrown exception back to the caller of our method**
- **The method header includes the reserved word *throws***
  - This means somewhere in this code an exception can occur
  - If it occurs this method **will not** deal with it

```
public int getAge( ) throws InputMismatchException {
    . . .
    int age = scanner.nextInt( );
    . . .
    return age;
}
```

Propagating Exceptions

- **Instead of handling the exception this method has decided to throw the exception as well. {someone else will handle it}**
- **The exception is propagated up the call stack as the caller may also throw the exception to its caller**

- **Note:**
  - At some point the exception should be caught and handled
  - The exception cannot be avoided indefinitely
  - Therefore there must be some method that can handle the exception or a crash may occur

Throwing Exceptions

- We can write a method that throws an exception directly, i.e., this method is the origin of the exception
- Use the *throw* reserved to create a new instance of the exception or its subclasses
- The method header includes the reserved word *throws*

```
public void doWork(int num) throws Exception {
    . . .
    if (num != val) throw new Exception("Invalid val");
    . . .
}
```

‘throws’ keyword

```java
public class djfjee {

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        method1();
    }

    static void method1() throws Exception{
        method2();
    }

    static void method2() throws Exception{
        throw new Exception();

    }
}
```

Exception not caught and handled so program crashes

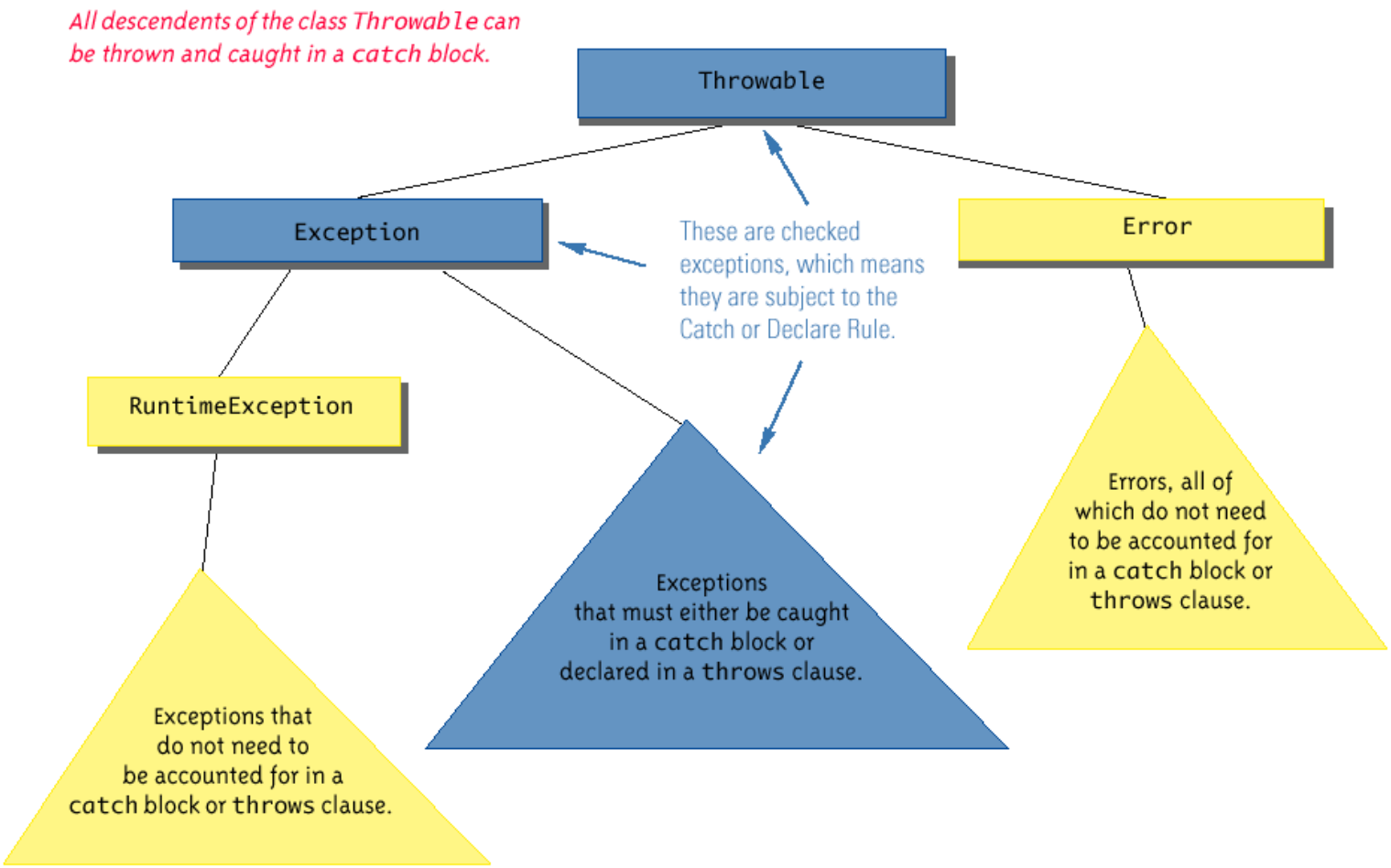
```
<terminated> djfjee [Java Application] C:₩Program Files₩Java₩jre7₩
Exception in thread "main" java.lang.Exception
        at classTest.djfjee.method2(djfjee.java:26)
        at classTest.djfjee.method1(djfjee.java:21)
        at classTest.djfjee.main(djfjee.java:14)
```

Exception Types

- All types of thrown errors are instances of the *Throwable* class or its subclasses
- Serious errors are represented by instances of the *Error* class or its subclasses
- Exceptional cases that common applications should handle are represented by instances of the *Exception* class or its subclasses

Exception Types

Display 9.10 **Hierarchy of Throwable Objects**



Catch or Declare Rule

- **Most ordinary exceptions might be thrown within a method must be accounted for in one of two ways:**
  1. The code that can throw an exception is placed within a ***try*** block, and the possible exception is caught in a ***catch*** block within the same method
  2. The possible exception can be declared at the start of the method definition by placing the exception class name in a ***throws*** clause

Catch or Declare Rule (Contd)

- The first technique handles an exception is a *catch* block
- The second technique is a way to shift the exception handling responsibility to the method that invoked the exception throwing method
- The invoking method must handle the exception, unless it too uses the same technique to "pass the buck"

Catch or Declare Rule (Contd)

- **In any one method, both techniques can be mixed**
  - Some exceptions may be caught, and others may be declared in a *throws* clause
- **However, these techniques must be used consistently with a given exception**
  - If an exception is not declared, then it must be handled within the method
  - If an exception is declared, then the responsibility for handling it is shifted to some other calling method
  - Note that if a first method definition encloses an invocation of a second method, and the second method can thrown an exception and does not catch it, then the first method must catch or declare it

Checked and Unchecked Exceptions

- **Exceptions that are subject to the catch or declare rule are called *checked* exceptions**
  - The compiler checks to see if they are accounted for with either a catch block or a throws clause
  - The classes ***Throwable***, ***Exception***, and all descendants of the class ***Exception*** are checked exceptions (Except RuntimeException and Derived classes of it)
- **All other exceptions are *unchecked* exceptions**
- **The class *Error* and all its descendant classes are called *error classes***
  - *Error classes* are *not* subject to the Catch or Declare rule

Exceptions to the rule

- **Checked exceptions must follow the Catch or Declare rule**
  - Programs in which these exceptions can be thrown will not compile until they are handled properly
- **Unchecked exceptions are exempt from the Catch or Declare Rule**
  - Programs in which these exceptions are thrown simply need to be corrected, as they result from some sort of error

Exceptions and Inheritance

- **When a method in a derived class is overridden, it should have the same exception classes listed in its *throws* clause that it had in the base class**
  - Or it should have a subset of them
- **A derived class may not add any exceptions to the *throws* clause**
  - But it can delete some

When to use Exceptions

- **Exceptions should be reserved for situations where a method encounters *an unusual or unexpected case that cannot be handled easily in some other way***
- **When exception handling must be used, here are some basic guidelines:**
  - Include ***throw*** statements and list the exception classes in a ***throws*** clause within a method definition
  - Place the ***try*** and ***catch*** blocks in a different method

When to use Exceptions (Contd)

- **Here is an example of a method from which the exception originates:**

```
public void someMethod() throws SomeException
{
        ...
        throw new SomeException(SomeArgument);
        ...
}
```

When to use Exceptions (Contd)

- **When *someMethod* is used by an *otherMethod*, the *otherMethod* must then deal with the exception:**

```
public void otherMethod()
{
        try
        {
                someMethod();
                ...
        }
        catch(SomeException e)
        {
                CodeToHandleException
        }
        ...
}
```

Exception Controlled Loops

- **Sometimes it is better to simply loop through an action again when an exception is thrown, as follows:**

```
boolean done = false;
while(!done)
{
    try
    {
        CodeThatMayThrowAnException
        done = true;
    }
    catch (SomeExceptionClass e)
    {
        SomeMoreCode
    }
}
```

Exception Controlled Loops

Display 9.11 **An Exception Controlled Loop**

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class InputMismatchExceptionDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number = 0; //to keep compiler happy
        boolean done = false;
```

(continued)

Exception Controlled Loops (Contd)

Display 9.11 An Exception Controlled Loop

```
        while (! done)
        {
            try
            {
                System.out.println("Enter a whole number:");
                number = keyboard.nextInt();
                done = true;
            }
            catch(InputMismatchException e)
            {
                keyboard.nextLine();
                System.out.println("Not a correctly written whole number.");
                System.out.println("Try again.");
            }
        }

        System.out.println("You entered " + number);
    }
}
```

If `nextInt` *throws an exception, the* `try` *block ends and so the* `boolean` *variable* `done` *is not set to* `true`.

(continued)

Exception Controlled Loops (Contd)

Display 9.11 **An Exception Controlled Loop**

SAMPLE DIALOGUE

```
Enter a whole number:
forty two
Not a correctly written whole number.
Try again.
Enter a whole number:
fortytwo
Not a correctly written whole number.
Try again.
Enter a whole number:
42
You entered 42
```

Self-Test (2)

- **Exception을 발생시키는 ExceptionDemo 클래스를 작성할 것**
  - ExceptionDemo 클래스에는 사용자로부터 정수 값을 받는 main 메소드와 입력 값에 따라 exception을 발생시키는 메소드인 exerciseMethod가 있음
  - ExceptionDemo 프로그램을 수행 시 다음과 같이 출력되도록 ExceptionDemo 클래스와 NegativeNumberException을 수정할 것
    - 입력 값: 양의 정수
      - In finally block
      - Exception is caught in main
    - 입력 값: 음의 정수
      - This number cannot be accepted!!
      - Exception is caught in exerciseMethod
      - In finally block
      - After finally block
    - 입력 값: 0
      - No Exception
      - In finally block
      - After finally block
    - 입력 값: 808
      - End of loop

Self-Test (2) (Contd)

- **양수 값을 입력할 경우 기존에 자바에 정의되어 있는 Exception이 발생**
  - exerciseMethod는 해당 Exception을 main method로 propagate 함.
  - main method는 해당 Exception을 처리해야 함.
- **NegativeNumberException은 Exception 클래스를 extends 하는 프로그래머 정의 exception 클래스**
- **NegativeNumberException 클래스의 구성은 프로그래머 정의 exception 클래스 권고 사항에 따라 2개의 생성자를 작성해야 함**
  - This number cannot be accepted!! 출력은 getMessage() 메소드를 통해 이루어져야 함.

Self-Test (2) (Contd)

- 프로그램 수행 예시

Problems @ Javadoc Declaration Console

```
<terminated> ExceptionDemo [Java Application] C:₩Program
100
In finally block
Exception is caught in main
-100
This number cannot be accepted!!
Exception is caught in exerciseMethod
In finally block
After finally block
0
No Exception
In finally block
After finally block
808
End of loop
```