Software Practice 1 - Error Handling

- Exception
- Exception Hierarchy
- Catching Exception
- Userdefined Exception
- Practice#5

Prof. Joonwon Lee

T.A. Jaehyun Song
Jongseok Kim (42)

T.A. Sujin Oh
Junseong Lee (43)

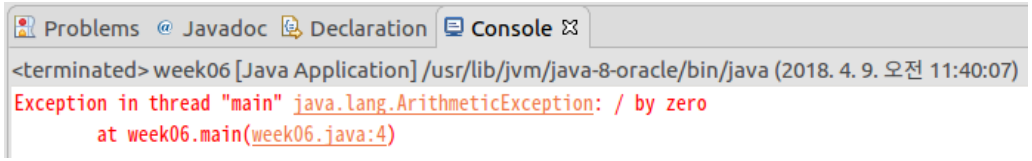
Exception

- An exception (or exceptional event) is a problem that arises during the execution of a program
- When an **Exception occurs** the normal flow of the program is disrupted and the **program/Application terminates abnormally**, which is not recommended, therefore, these **exceptions are to be handled**

Exception – occurring scenarios

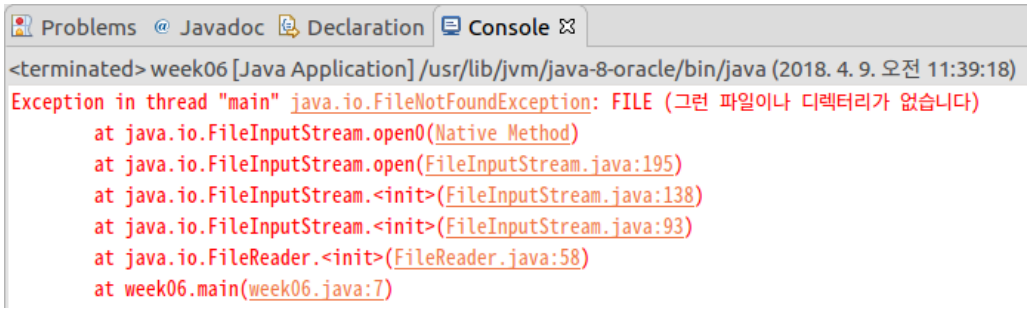
$Enter a integer number : 4.05

// A user has entered an invalid data



open("404_file","w")

// A file needs to be opened cannot be found



listen(channel_B)

// channel_B is disconnected

Exception – Three Categories

- **Checked exceptions**
  - Compile time exceptions
  - Programmer should take care of (handle) these exceptions
- **Unchecked exceptions**
  - Runtime exceptions
  - Including programming bugs, logic errors or improper use of an API
- **Errors**
  - These are not exception at all, but problems arise beyond the control of the user and programmer
  - e.g. OutofMemoryError, ThreadDeath

Checked Exceptions

- If you use **FileReader** class to read data from a file, if the file specified in its constructor doesn't exist, then a ***FileNotFoundException*** occurs, and the compiler prompts the programmer to handle the exception.

```
import java.io.File;
import java.io.FileReader;
public class FilenotFound_Demo {
        public static void main(String args[]) {
                File file = new File("E://file.txt");
                FileReader fr = new FileReader(file);
        }
}
```

```
C:\>javac FilenotFound_Demo.java
FilenotFound_Demo.java:8: error: unreported exception FileNotFoundException;
must be caught or declared to be thrown FileReader fr = new FileReader(file); ^ 1 error
```

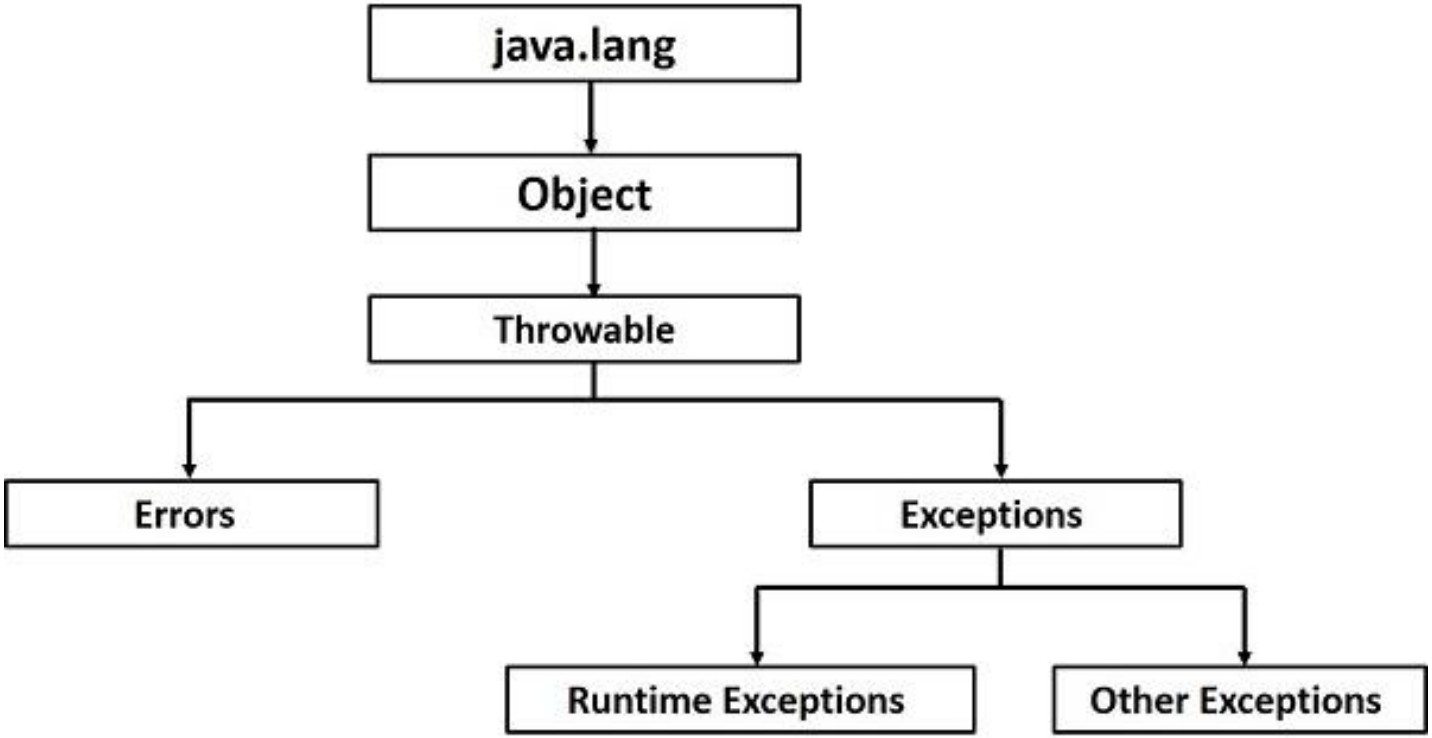
Unchecked Exceptions

- If you have declared an array of size and trying to call the 6th element of the array then an ***ArrayIndexOutOfBoundsException*** *exception* occurs.

```
public class Unchecked_Demo {
        public static void main(String args[]) {
                int num[] = {1, 2, 3, 4};
                System.out.println(num[5]);
        }
}
```

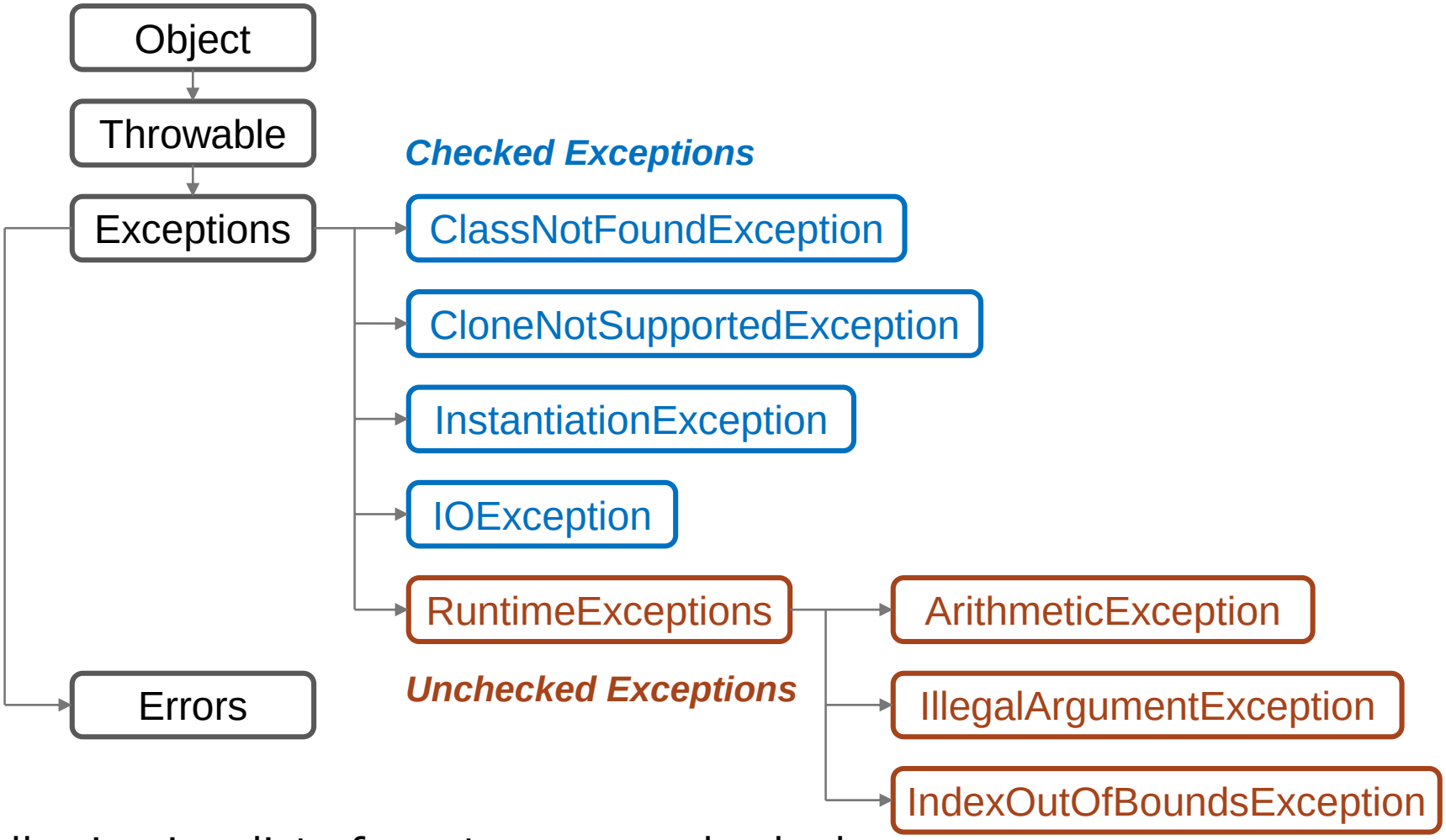
```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at Exceptions.Unchecked_Demo.main(Unchecked_Demo.java:8)
```

Exception Hierarchy



Following is a list of most common checked and unchecked Java's Built-in Exceptions.

Exception Hierarchy



Following is a list of most common checked and unchecked Java's Built-in Exceptions.

Exception Methods

Sr.No.	Method & Description
1	public String getMessage() Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
2	public Throwable getCause() Returns the cause of the exception as represented by a Throwable object.
3	public String toString() Returns the name of the class concatenated with the result of getMessage().
4	public void printStackTrace() Prints the result of toString() along with the stack trace to System.err, the error output stream.
5	public StackTraceElement [] getStackTrace() Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
6	public Throwable fillInStackTrace() Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

Catching Exceptions

- A method catches an exception using a combination of the **try** and **catch** keywords
- A try/catch block is placed around the code that might generate an exception

```
try {
        // Protected code
} catch(ExceptionName e1) {
        // Catch block
}
```

try-catch Example

```
// File Name : ExcepTest.java
import java.io.*;
public class ExcepTest {
        public static void main(String args[]) {
                try {
                        int a[] = new int[2];
                        System.out.println("Access element three :" + a[3]);
                } catch(ArrayIndexOutOfBoundsException e) {
                        System.out.println("Exception thrown :" + e);
                }
                System.out.println("Out of the block");
        }
}
```

Multiple Catch Blocks

```
try {
        // Protected code
} catch(ExceptionType1 e1) {
        // Catch block
} catch(ExceptionType2 e2) {
        // Catch block
} catch(ExceptionType3 e3) {
        // Catch block
}
```

The Finally Block

- The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.

```
try {
            // Protected code
} catch(ExceptionType1 e1) {
            // Catch block
} catch(ExceptionType2 e2) {
            // Catch block
} catch(ExceptionType3 e3) {
            // Catch block
} finally {
            //The finally block always executes.
}
```

The Finally Block Example

```
public class ExcepTest {
        public static void main(String args[]) {
                int a[] = new int[2];
                try {
                        System.out.println("Access element three :" + a[3]);
                } catch(ArrayIndexOutOfBoundsException e) {
                        System.out.println("Exception thrown :" + e);
                } finally {
                        a[0] = 6;
                        System.out.println("First element value: " + a[0]);
                        System.out.println("The finally statement is executed");
                }
        }
}
```

Notes

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses whenever a try/catch block is present.
- The try block cannot be present without either catch clause or finally clause.
- Any code cannot be present in between the try, catch, finally blocks.

The Throws/Throw Keyword

- If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.

```
import java.io.*;
public class className {
        public void deposit(double amount) throws RemoteException {
                // Method implementation
                throw new RemoteException();
        }
        // Remainder of class definition
}
```

The try-with-resources

- Generally, when we use any resources like streams, connections, etc. we have to close them explicitly using finally block. In the following program, we are reading data from a file using **FileReader** and we are closing it using finally block.

```
try(FileReader fr = new FileReader("file path")) {
            // use the resource
} catch() {
            // body of catch
}
```

Example

```
import java.io.FileReader;
import java.io.IOException;
public class Try_withDemo {
        public static void main(String args[]) {
                try(FileReader fr = new FileReader("E://file.txt")) {
                        char [] a = new char[50];
                        fr.read(a); // reads the contentto the array
                        for(char c : a)
                                //prints the characters one by one
                                System.out.print(c);
                } catch(IOException e) {
                        e.printStackTrace();
                }
        }
}
```

Notes

- To use a class with try-with-resources statement it should implement **AutoCloseable** interface and the **close()** method of it gets invoked automatically at runtime.
- You can declare more than one class in try-with-resources statement.
- While you declare multiple classes in the try block of try-with-resources statement these classes are closed in reverse order.
- Except the declaration of resources within the parenthesis everything is the same as normal try/catch block of a try block.
- The resource declared in try gets instantiated just before the start of the try-block.
- The resource declared at the try block is implicitly declared as final.

User-defined Exception

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class.

```
class MyException extends Exception { }
```

Example

```
// File Name InsufficientFundsException.java
import java.io.*;
public class InsufficientFundsException extends Exception {
        private double amount;
        public InsufficientFundsException(double amount) {
                this.amount = amount;
        }
        public double getAmount() {
                return amount;
        }
}
```

Example (cont'd)

```
// File Name CheckingAccount.java
import java.io.*;
public class CheckingAccount {
        private double balance;
        private int number;
        public CheckingAccount(int number) {
                this.number = number;
        }
        public void deposit(double amount) {
                balance += amount;
        }
        public void withdraw(double amount) throws InsufficientFundsException {
                if(amount <= balance) {
                        balance -= amount;
                }
                else {
                        double needs = amount - balance;
                        throw new InsufficientFundsException(needs);
                }
        }
        public double getBalance() {
                return balance;
        }
        public int getNumber() {
                return number;
        }
}
```

Example (cont'd)

```
// File Name BankDemo.java
public class BankDemo {
        public static void main(String [] args) {
                CheckingAccount c = new CheckingAccount(101);
                System.out.println("Depositing $500...");
                c.deposit(500.00);
                try {
                        System.out.println("Withdrawing $100...");
                        c.withdraw(100.00);
                        System.out.println("Withdrawing $600...");
                        c.withdraw(600.00);
                } catch(InsufficientFundsException e) {
                        System.out.println("Sorry, but you are short $" + e.getAmount());
                        e.printStackTrace();
                }
        }
}
```

[Lab – Practice #5]

- **Implement SimpleCalculator.java**
- **When methods try to illegal action**
  - Throw exception (define exception when it doesn't exist)
  - After throw exception, process should be aborted
- **Exceptions**
  - OutOfRangeException // Range of result (0 ~ 9999)
  - AddZeroException // add() is not allowed to add zero
  - SubtractZeroException // subtract() is not allowed to subtract zero

[Lab – Practice #5]

- **class SimpleCalculator**
  - Field
    - result // integer number (0~9999)
  - Methods :
    - add(int value1, int value2)
    - subtract(int value1, int value2)
    - print() // print result
    - reset() // reset result as 0

[Lab – Practice #5]

- **Exceptions**
  - Field
    - String message
  - Methods/constructor
    - Exceptions()
    - Exceptions(String message)
    - getMessage()

[Lab – Practice #5]

- **File “week06.java” can has only week06 class**
- **Functions in “week06.java” can NOT have any *throws* keyword**
- **If method throw exception in scenario, handle the exception with “try-catch”**
  - Print which exception is thrown
  - Hint : use a message in exception
- **Print result of each scenario**

[Lab – Practice #5]

- **Addition example**

```
0. Exit
1. Addition
2. Subtract
3. Print Result
4. Reset
1
Input value 1:
10000
Input value 2:
0
#### ERROR ####
Range of input is 0~9999
```

```
0. Exit
1. Addition
2. Subtract
3. Print Result
4. Reset
1
Input value 1:
100
Input value 2:
0
#### ERROR ####
Add is not allowed to add zero
```

[Lab – Practice #5]

- **Subtract example**

```
0. Exit
1. Addition
2. Subtract
3. Print Result
4. Reset
2
Input value 1:
999
Input value 2:
8
0. Exit
1. Addition
2. Subtract
3. Print Result
4. Reset
3
Result: 991
```

```
0. Exit
1. Addition
2. Subtract
3. Print Result
4. Reset
2
Input value 1:
9
Input value 2:
99999
#### ERROR ####
Range of input is 0~9999
```

[Lab – Practice #5]

- **Other example**

```
0. Exit
1. Addition
2. Subtract
3. Print Result
4. Reset
4
0. Exit
1. Addition
2. Subtract
3. Print Result
4. Reset
3
Result: 0
```

[Submit]

- **Upload to i-Campus**
  - Compress your SimpleCalculator.java & week06.java file to zip file
  - Do not use any package keyword
  - File name: studentID_lab04.zip

- **Due date**
  - Today 23:59:59
    - Class 42 (04/09 Monday)
    - Class 43 (04/11 Wednesday)
  - Penalty: -10% of each lab score per one day