COMP90041
Programming and Software Development
2020 - Semester 1
Lab 9 - Week 10

Yuhao(Beacon) Song

yuhsong1@unimelb.edu.au

Introduction

- Timetable
  - Tue(11) 14:15-15:15 (Melbourne Time) Join URL: https://unimelb.zoom.us/j/490084146
  - Tue(07) 16:15-17.15 (Melbourne Time) Join URL: https://unimelb.zoom.us/j/291505765

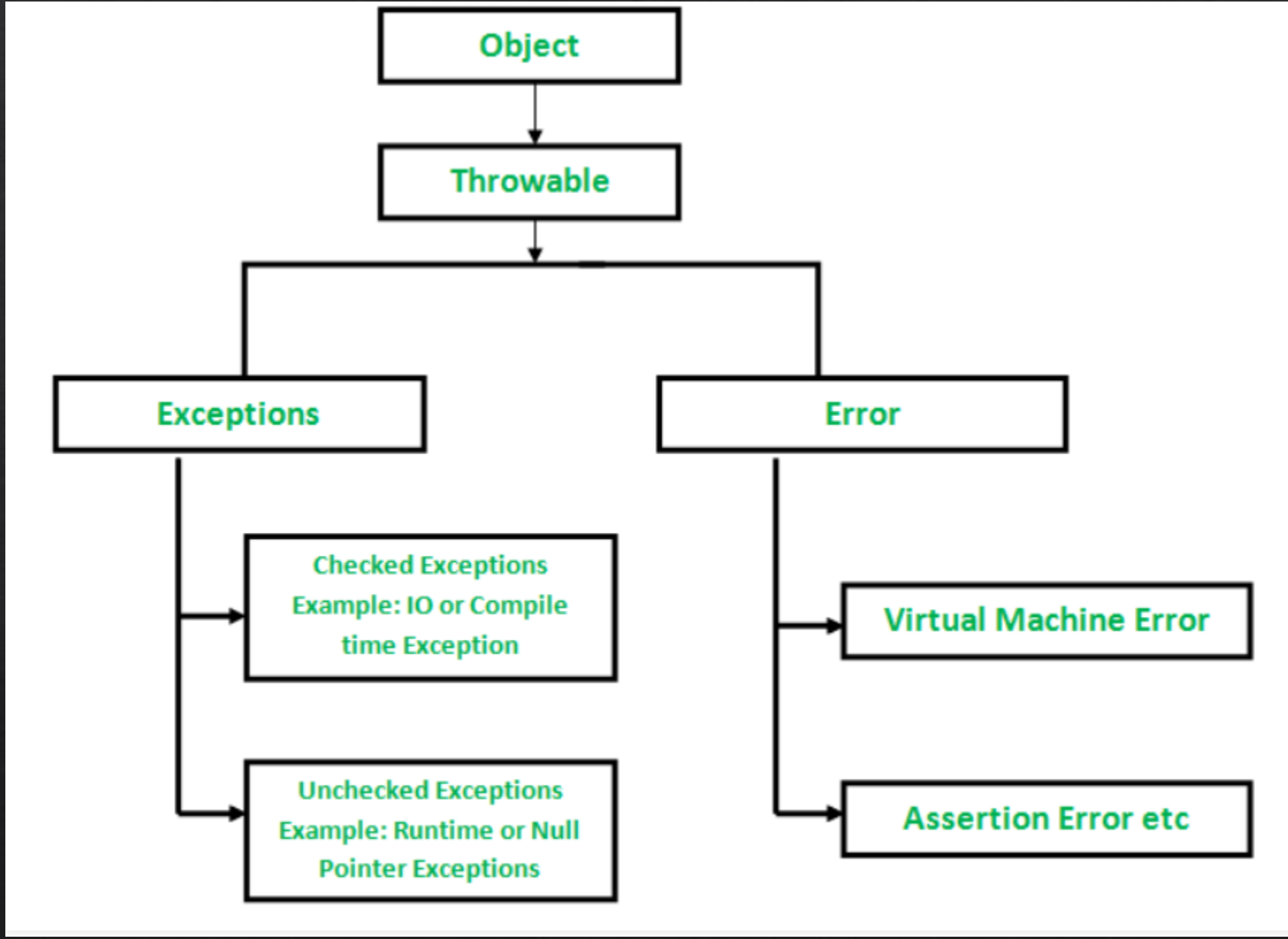
- Contact
  - yuhsong1@unimelb.edu.au
  - yuhsong@student.unimelb.edu.au
  - GitHub: https://github.com/Beaconsyh08/COMP90041_Programming_and_Software_Development_Tutorials.git

Outline

- Lecture Review
- Exercise

Exception

- **Checked Exceptions**
  - **Must** be handled.
- **Unchecked Exceptions**
  - Could be handled



Exception Throwing & Handling

- **Throwing an Exception:** Java library software (or programmer-defined code) provides a mechanism that signals when something unusual happens

- **Handling the Exception:** In another place in the program, the programmer must provide code that deals with the exceptional case

Five Keywords

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It can be followed by finally block later. It must be preceded by try block which means we can't use catch block alone.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

(https://www.javatpoint.com/exception-handling-in-java)

The try-throw-catch Trio

```
try {
    throw new DemoMyException();  // throws the exception
    System.out.println("Can not go here");
} catch (DemoMyException e) {  // Catch the More Specific
    e.printStackTrace();  // Useful for debugging, but not
    System.out.println(e.getMessage());
} catch (Exception e) {
    System.out.println("Catch any exception");
} finally {
    System.out.println("But you can go here (optional)");
}
System.out.println("finish!");
```

try

- Two cases:
- **No exception is thrown** in the **try** block
  - The code in the try block is executed to the end of the block
  - The catch block is skipped
  - The execution continues with the code placed after the catch block
- An **exception is thrown** in the **try** block and caught in the **catch** block
  - The rest of the code in the try block is skipped
  - Control is transferred to a following catch block (in simple cases)
  - The thrown object is plugged in for the catch block parameter
  - The code in the catch block is executed
  - The code that follows that catch block is executed (if any)

catch

- When an exception is thrown, the catch block begins execution
  - The catch block has only one parameter
  - The exception object thrown is plugged in for the catch block parameter
- The execution of the catch block is called catching the exception, or handling the exception; the **catch block** is an exception handler
  - Whenever an exception is thrown, it should **ultimately be handled/caught** by some **catch block**

```
catch (myException e) {
ExceptionHandlingCode}
```

- **e** is called the **catch block parameter**
  - It specifies the **type** of thrown exception object that the catch block can catch
  - It provides a **name** (for the thrown object that is caught) on which it can operate in the catch block
  - The identifier **e** is often used by convention, but any non-keyword identifier can be used

Exception Handling: Two ways

1. **Throw** an exception is placed **within a try block**, and the possible exception is caught in a **catch block** within **the same method -- throw**

- The first technique handles an exception in a **catch block**

2. **Declared** at the **start of the method definition** by placing the exception class name in a **throws clause -- throws**

- 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 **shift the responsibility** as well

- Ultimately, every exception that is thrown **must eventually** be caught/handled by a **catch block** in some method that does not just declare the exception class in a throws clause

throw

throw *new ExceptionClassName(PossiblySomeArguments);*

- If exception is thrown, try block stops
  - Normally, the flow of control is transferred to another portion of code known as **the catch block**

- The value thrown is the argument to the throw operator, and is always an object of some exception class

throws

public void aMethod() ***throws*** *AnException*

- If a method can throw an exception but does not catch it, it must provide a **warning** in the heading
  - This warning is called a **throws clause**
  - The process of including an exception class in a throws clause is called **declaring the exception**

- If a method throws an exception and does not catch it, then the method invocation ends immediately

```
public static void myMehtod() throws DemoMyException {
    throw new DemoMyException();
    System.out.println("Can not go here");
}
```

Programmer-Defined Exception Class Guidelines

- Must be a **derived class** of an already existing exception class
- **At least two constructors** should be defined, sometimes more
  - A **no-argument constructor** that includes a call to super with a default string as the argument
  - A **constructor** that takes a string argument and begins with a call to super, which takes the string argument

```
public class DemoMyException extends Exception{
    public DemoMyException() {
        super("Default My Exception Message");
    }

    public DemoMyException(String message) { super(message); }
}
```

Preserve getMessage

- The method getMessage is inherited
- 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 behaviour must be preserved in all programmer defined exception class, **two constructors must be included by convention**

Tip: 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 handling Exercise

1. Write a Java program that prompts the user for two integers. Use a try/catch block to handle the InputMismatchException.

2. Define an Exception class called NegativeNumberException. The class should have a constructor with no parameters. If an exception is thrown with this zero-argument constructor, the getMessage() method should return "Negative Number Not Allowed!" This class should also have a construction with a single parameter of type String. If an exception is thrown with this construction, then the getMessage() method returns the value that was used as an argument to the constructor.

3. Revise the program in Exercise 1 above to throw a NegativeNumberException if the user enters a negative number.

4. Try to Put it in a while-loop and ask the user input again if exception happen