Java Programming

Exception Handling

ISTAD

Contents

1. Introduction to Exception
2. Exception Hierarchy
3. Catching and Handling Exception
4. Try-with-Resource
5. How to throw Exception
6. Custom Exception

Introduction to Exception

- Exception is an event (or problem) that occurs during the execution of a program
- An exception can occur because:
  - A user has entered **invalid data**
  - A file needing to be opened **cannot be found**
  - A network **connection has been lost** in the middle of communications
  - `null.someMethod();`
- When an exception occurs, the program will be forced to terminate.

Introduction to Exception

- Traditionally, we can use if clause to avoid error:

```
public static void main(String[] args) {
    int[] nums = {2, 4, 6};
    for(int i=0; i<5; i++){
        if(i >= nums.length)
            break;
        else
            System.out.println(nums[i]);
    }
}
```

Now, we can use modern way: exception handling

Introduction to Exception

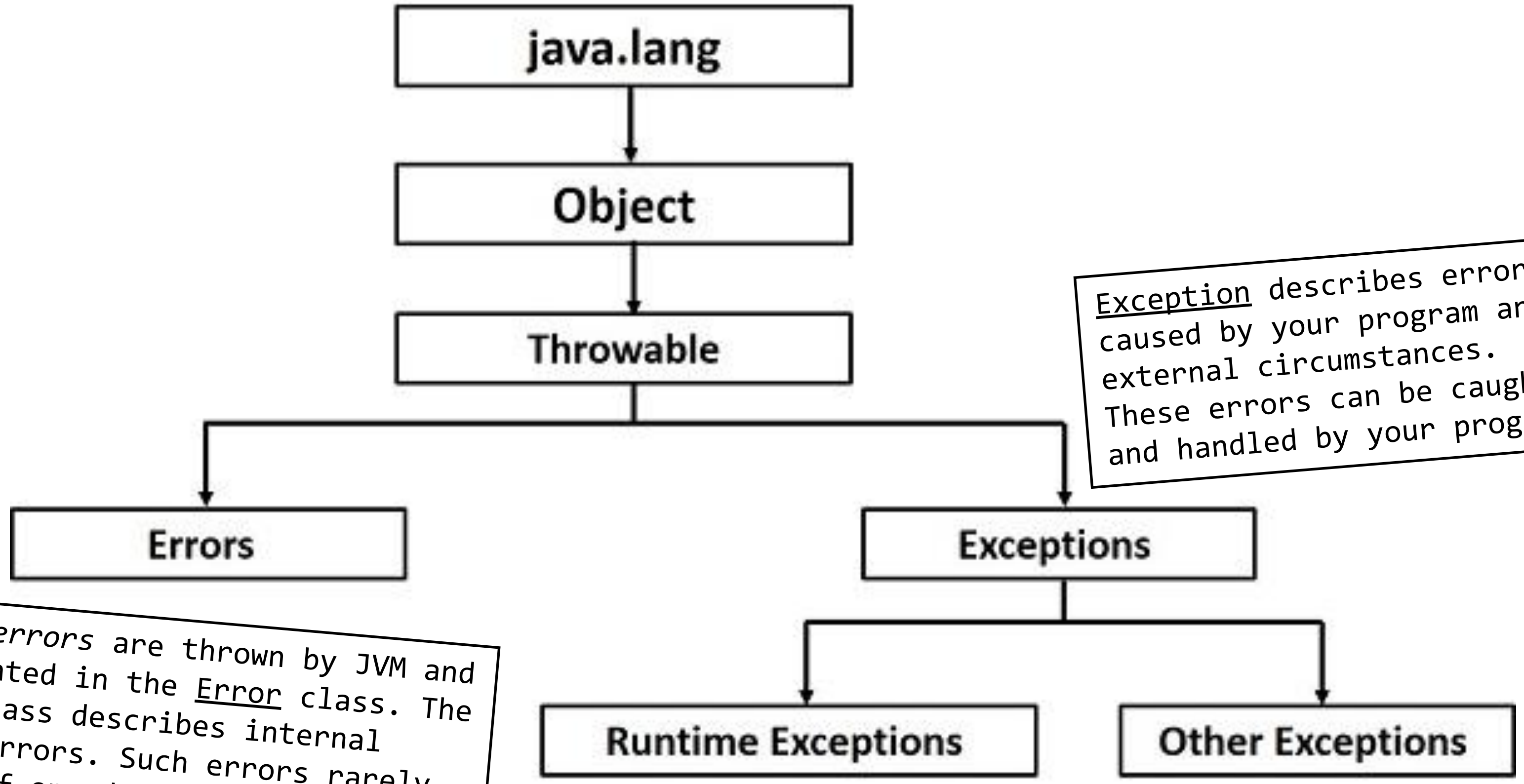
What is Exception Handling?

- Is an mechanism for handling exception by detecting and responding to exceptions in a systematic manner

Advantages of Handling Exception

- Separating Error-Handling code from “Regular” code
- Propagating Errors Up the call stack
- Grouping and differentiating error types

Exception Hierarchy



Exception describes errors caused by your program and external circumstances. These errors can be caught and handled by your program.

System errors are thrown by JVM and represented in the Error class. The Error class describes internal system errors. Such errors rarely occur. If one does, there is little you can do beyond notifying the user and trying to terminate the program gracefully.

Exception Hierarchy

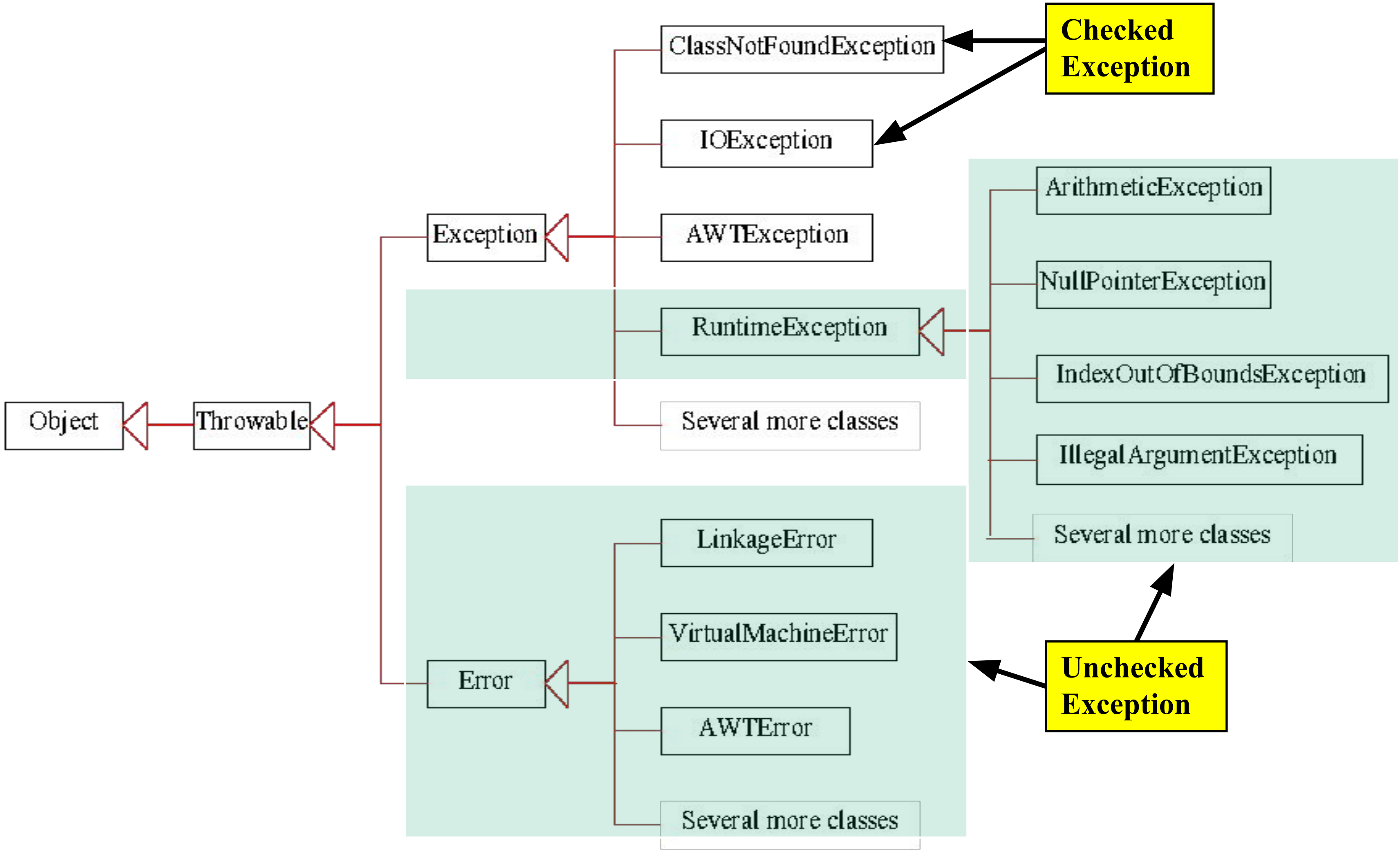
- All exception classes are subclass of the Exception class
- The Exception class is a sub classes of the Throwable class
- There is another sub class called “Error” which is also derived from the Throwable class.
- Errors:
  - An error occurs when a **dynamic linking failure** or other **hardware failure** in the JVM.
  - They are typically not handled by regular programs.
- Exception

  - An Exception indicates that a problem occurred, but it not a serious system problem.

Exception Hierarchy

- There are mainly two type of exceptions: **checked** and **unchecked** where error is considered as unchecked exception.
- Checked Exception: are kinds of exception that require developer to handle before compile source code. The compile expected that there will be error occur during execution. Example: FileNotFoundException, IOException, SQLException, …
- Unchecked Exception: are kinds of exception that the compiler doesn’t expected that there will be occured or not, so programmer no need to handle the error compile source code. Example: ClassCastException, NumberFormatException, ArithmeticException, …

Exception Hierarchy



Catching and Handling Exception

Try Block

- The code that might throw an exception should be write in Try Block
- Syntax

```
try {
   // code
}
// catch and finally blocks
```

- When an exception occurs in Try Block, it will throw an exception object
- Exception object will be caught by exception handler.

Catching and Handling Exception

Catch Block

- Block for write code to solve problem when exception appear.
- Each *catch block* is an *exception handler*, and it handles the type of exception by its argument
- Syntax

```
try {
    // code
} catch (ExceptionType1 arg) {
    // code
} catch (ExceptionType2 arg) {
    // catch and finally blocks
}
```

Catching and Handling Exception

Catch Block

- Catch Block contains code that is executed when the exception is caught
- No code between end of **Try Block** and start of **Catch Block**
- **ExceptionType** must be the **Name of Class** that inherits from the **Throwable Class**
- A single catch block can handle more than one type of Exception
- Example:

```
catch (IOException | SQLException e) {
     System.out.println(e);
}
```

Catching and Handling Exception

Finally Block

- A finally block of code always executes, irrespective of occurrence of an Exception
- Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code (try/block).

```
try {
    // code
} finally {
    // code
}
```

Catching and Handling Exception

Finally Block

```
public static void main(String[] args) {
    BufferedReader bf = null;
       try{
           bf = new BufferedReader(new FileReader("C:\\person.txt"));
       } catch(FileNotFoundException ex){
           //code
       } catch(IOException ex){
           //code
       } finally{
           //code
           if(bf! = null) {
                  try {
                      bf.close(); //close the file if it is opened
                  } catch (IOException e) { //code }
           }}}
```

Catching and Handling Exception

Putting all together

```java
public static void main(String []args){
    InputStreamReader reader = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(reader);
    try{
        System.out.print("Enter num: ");
        int n = Integer.parseInt(br.readLine());
    } catch(IOException e){
        System.out.println(e.getMessage());
    } catch (NumberFormatException e) {
        System.out.println(e.getMessage());
    } finally{
        System.out.println("finally executed");
        System.out.println("although try block process");
    }}
```

Try-with-Resource

- JDK 7 introduces a new version of try statement known as try-with-resource statement
- Syntax:

```
try (resource-specification) {
   //use the resource
} catch (Exception e) {
   // code
}
```

Try-with-Resource

- The try-with-resource statement is a try statement that declares one or more resources
- Any object that implements from *java.lang.AutoCloseable* or *java.io.Closeable* can be passed as a parameter to try statement
- A resource is an object that used in program and must be closed after the program is finished
- The try-with-resource statement ensures that each resource is closed at the end of the statement, you do not have to explicitly close the resources.

Try-with-Resource

Java try-with-resource benefits:

- More readable code and easy to write
- Automatic resource management
- Number of lines of code is reduced
- No need of finally block just to close the resource
- Can open multiple resources in try-with-resources statement separated by a semicolon (;)
- When multiple resource are opened, it closes them in the reverse order to avoid any dependency issue.

Try-with-Resource

This an example of using with try-block statement

```java
public static void main(String[] args) throws Exception {
    try {
        BufferedReader br = new BufferedReader(new FileReader("src/file.txt"));
        System.out.println(br.readLine());
        br.close();
    } catch (FileNotFoundException e) {
        System.out.println(e.getMessage());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

in above example, we need to explicit call **br.close()** method to close BufferReader stream

Try-with-Resource

This an example of using with try-with-resource statement

```
public static void main(String[] args) throws Exception {
    try (BufferedReader br = new BufferedReader(new FileReader("src/file.txt"))) {
        System.out.println(br.readLine());
    } catch (FileNotFoundException e) {
        System.out.println(e.getMessage());
    } catch (IOException e) {
        e.printStackTrace();
    }}
```

in above example, we don’t need to explicit call **br.close()** method to close BufferReader stream.

How to throw exception

- The throw keyword is used to throw an exception explicitly.
- The throw statement requires a single argument: a ***throwable object***
- Only object of Throwable class or its subclass can be thrown
- Program execution stops on encounting throw statement, and the closet catch statement is checked for matching type of exception.
- Syntax:

```
throw Throwable_Instance;
```

- Example

```
throw new NullPointerException();
```

How to throw exception

Example of throw exception

```java
public class Main {
    public static void main(String[] args){
        useThrow(1, 0);
    }
    static void useThrow(int a, int b){
        if(b == 0)
            throw new ArithmeticException();
    }
}
```

How to throw exception

Throws keyword

- Throws is used when you don’t want this method to handle the exception
- Throws keyword is used to declare an exception
  - Apply throws keyword to method
  - Provide information to the caller about exception
- Give information to programmer that there may occur an exception so it is better to provide exception handling code.
- Syntax:

```
type methodName (parameterList) throws ExceptionList {
   // definition of method
}
```

How to throw exception

Example of thrown exception:

```
public static void main(String[] args){
    try {
        testFile();
    } catch (FileNotFoundException e) {
        System.out.println("File Not Found");
    } catch (IOException e) {
        System.out.println("Error with File");
    }
}

static void testFile() throws FileNotFoundException, IOException {
    BufferedReader br = new BufferedReader(new FileReader("src/file.txt"));
    System.out.println(br.readLine());
}
```

How to Custom Exception

- Java allows you to create your own exceptions that is called “Custom Exception” or “User-Defined Exception”
- Custom exceptions are used to customize the exception (error messages) according to developer need.
- Keep the following points in mind when writing user-defined exception:
  - All exceptions must be a child of Throwable
  - If you want to write a checked exception, you need to extend the Exception class
  - If you want to write a runtime exception, you need to extend the RuntimeException class

How to Custom Exception

Step to create custom exception

1. Create your exception class

```
classs UserDefinedException extends Exception {
    UserDefinedException() {
        // statement
    }
}
```

2. Raise Error

```
DataType methodName () throws UserDefinedException {
    // statement
    throw UserDefinedException_Instance;
}
```

How to Custom Exception

Example of thrown exception:

3. Catch Error

```
try {
    method that throws exception
} catch (UserDefinedException arg) {
    // statement
}
```

How to Custom Exception

Example of creating *UserDefinedException* class

```
public class ScoreFormatException extends Exception {
    public ScoreFormatException() {
        super();
    }

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

How to Custom Exception

Example of raising error

```
class Student {
    String name;
    double score;
    Student (String name, double score) throws ScoreFormatException{
        this.name = name;
        if(score < 0 || score > 100)
            throw new ScoreFormatException("Invalid score");
        else
            this.score = score;
    }
    public double getScore(){
        return score;
    }
}
```

How to Custom Exception

Example of catching error

```java
class Main{
    public static void main(String[] args){
        try {
            Student student = new Student("kaka", 120);
            System.out.println("Your Score is : " +
              student.getScore());
        } catch (ScoreFormatException e) {
            System.out.println(e);
        }
    }
}
```

Thanks

Be humble, and learn