08_2 Throwing & Catching Exceptions

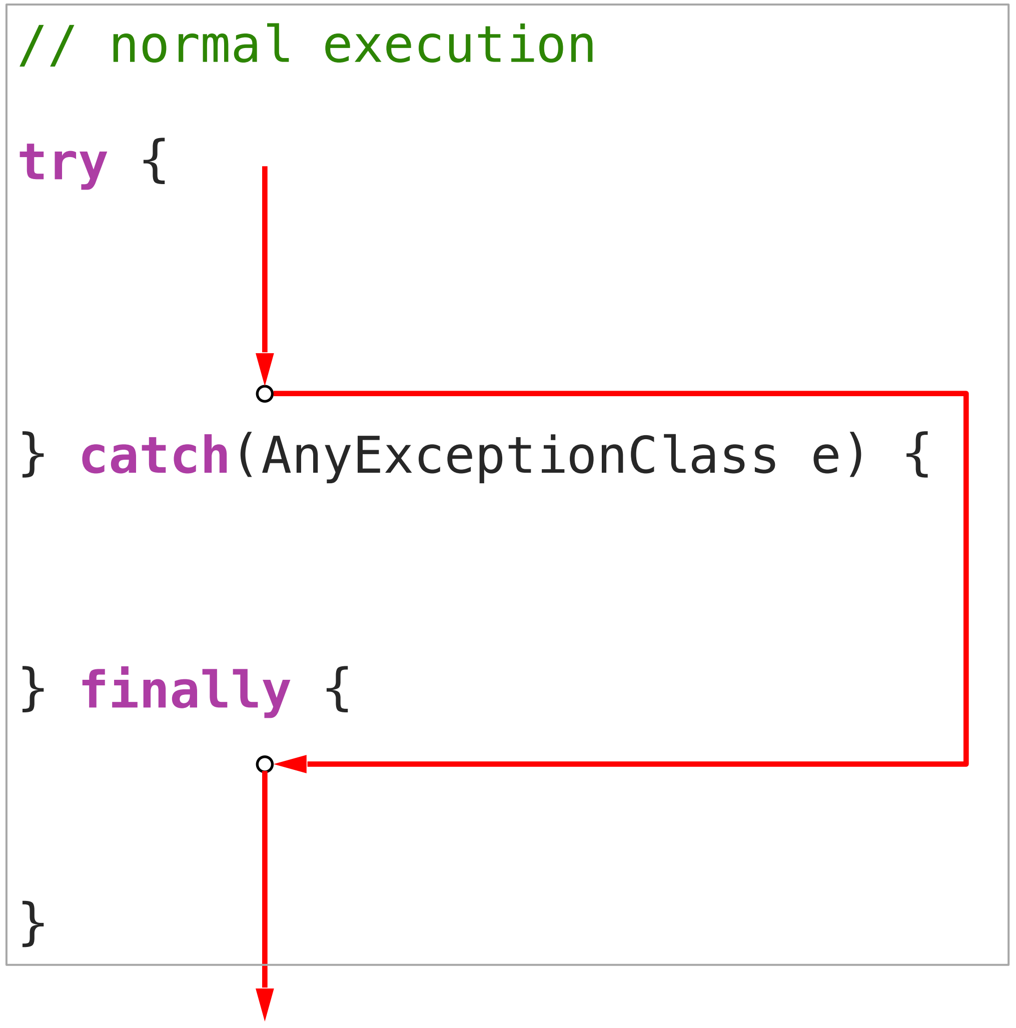
Object-Oriented Programming

Exception Handling Code

- try-catch-finally block

```
try {
    // Code that can throw an exception
}
catch (AnyExceptionClass e) {
    // Exception handling
}
finally {
    // Code that is always executed
    // whether the exception is thrown or not
}
```

Try-Catch-Finally Block



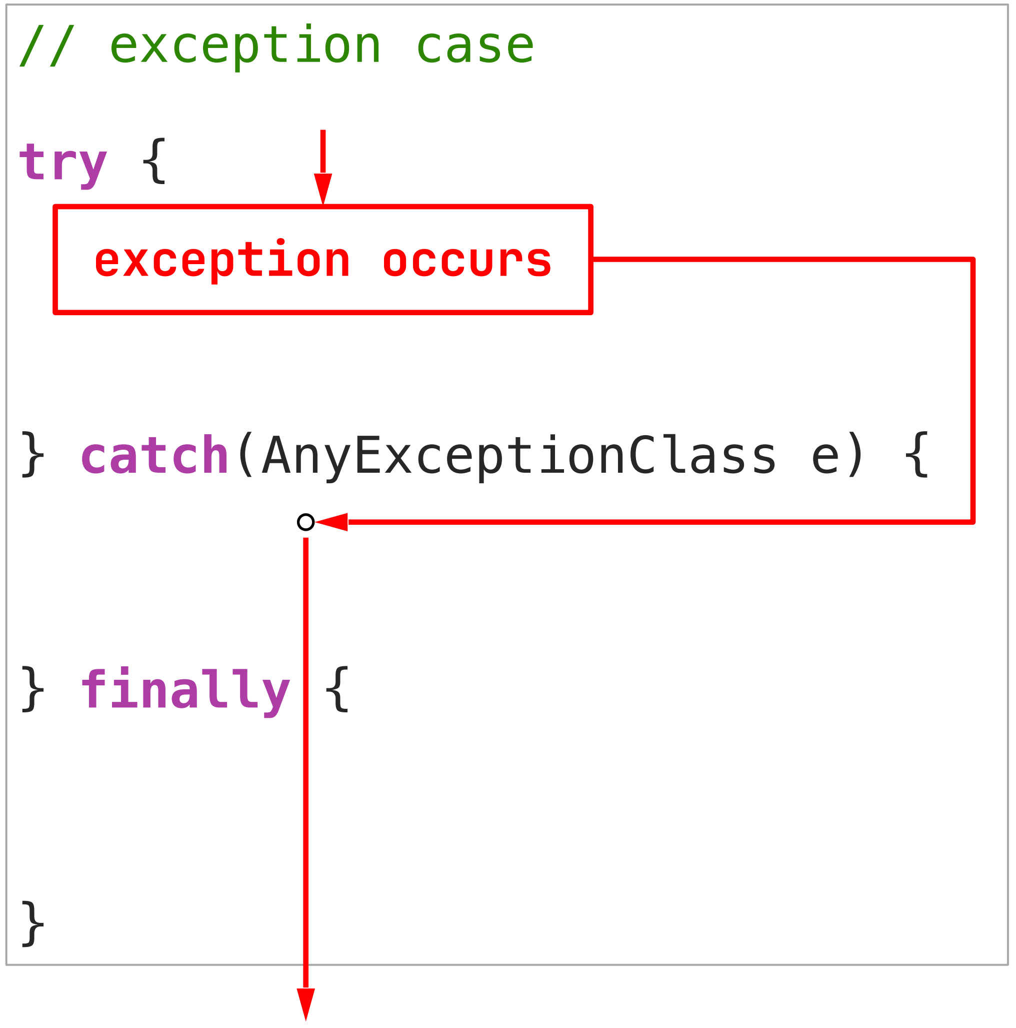
```
// normal execution

try {


} catch(AnyExceptionClass e) {


} finally {


}
```



```
// exception case

try {


} catch(AnyExceptionClass e) {


} finally {


}
```

Example: Flow in try-catch Block (1/2)

No Exception case

```
class ExceptionEx02 {
    public static void main(String args[]) {
        System.out.println(1);
        System.out.println(2);
        try {
            System.out.println(3);
            System.out.println(4);
        } catch (Exception e) {
            System.out.println(5);
        }
        System.out.println(6);
    }
}
```

```
Output:
   1
   2
   3
   4
   6
```

Example: Flow in try-catch Block (2/2)

Exception case

```
class ExceptionEx02 {
    public static void main(String args[]) {
        System.out.println(1);
        System.out.println(2);
        try {
            System.out.println(3);
            System.out.println(0/0);
            System.out.println(4);
        } catch (Exception e) {
              System.out.println(5);
        }
        System.out.println(6);
    }
}
```

Division by Zero

```
Output:
   1
   2
   3
   5
   6
```

Example: TryCatchDemo

```
public class TryCatchDemo {
    public static void main(String[] args) {

        String[] str = new String[]{"123", "45", "abc"};
        int[] a = new int[3];
        for (int i = 0; i < 4; i++) {
            try {
                a[i] = Integer.parseInt(str[i]);
                System.out.println("Index " + i + " parseInt done");
            } catch(ArrayIndexOutOfBoundsException e) {
                System.out.println("Array index exception at index " + i);
            } catch(NumberFormatException e) {
                System.out.println("Number format exception at index " + i);
            } catch(Exception e) {
                System.out.println("Other exception at index " + i);
            } finally {
                System.out.println("finally index " + i + " done");
            }
        }
    }
}
```

```
Index 0 parseInt done
finally index 0 done
Index 1 parseInt done
finally index 1 done
Number format exception at index 2
finally index 2 done
Array index exception at index 3
finally index 3 done
```

Exception Catching Order

- All exception classes are descendants of java.lang.Exception
- Descendant exception class (specific exception) must be caught first
- Otherwise, the parent will catch all exceptions

```
try {

}
catch (Exception e) {

}
catch (ArrayIndexOufOfBoundsException e) {

}
catch (NumberFormatException e) {

}
```

Catching all exceptions here

No chance to catch any exception

Throwing Exception

- Two choices for exception handling code inside a method:
  ① Use the try-catch block to handle the exception
  ② Just throw the exception to the place where the method was called

- For throwing exception: Use ‘throws’ keyword at the header of the method:

```
return_type method_name (parameters) throws SomeException1, SomeException2, ... {


}
```

Example: ThrowingExceptionDemo

```java
public class ThrowingExceptionDemo {
    public static void main(String[] args) {
        String name = "java.lang.String2";
        try {
            // get 'Class' object having 'name'
            Class classObject = findClass(name);
        } catch (ClassNotFoundException e) {
            System.out.println("No class having name: " + name);
        }
    }

    static Class findClass(String name) throws ClassNotFoundException {
        Class classObject = Class.forName(name);
        return classObject;
    }
}
```

```
OUTPUT:
No class having name: java.lang.String2
```

Throw command

- Intentionally throwing an exception using ‘throw’ command
- The exception can be handled inside the method or ‘throws’ it to parent

```java
public class ExceptionEx03 {
    public static void main(String args[]) {
        try {
            Exception e = new Exception("My Exception");
            throw e; // throw the exception
        }
        catch (Exception e) {
            System.out.println("Error message: " + e.getMessage());
            e.printStackTrace();
        }
        System.out.println("Program ended");
    }
}
```

Example: ExceptionEx04

```
public class ExceptionEx04 {
    public static void main(String[] args) {
        try {
            method1();
            System.out.println(6);
        } catch (Exception e) {
            System.out.println(7);
        }
    }
```

```
    static void method1() throws Exception {
        try {
            method2();
            System.out.println(1);
        } catch (NullPointerException e) {
            System.out.println(2);
            throw e;   // rethrow the exception
        } catch (Exception e) {
            System.out.println(3);
        } finally {
            System.out.println(4);
        }
        System.out.println(5);
    }
    static void method2() throws NullPointerException {
        throw new NullPointerException();
    }
}
```

```
OUTPUT:
2
4
7
```

User Defined Exception

- Custom exception class defined by programmer
- To handle exceptions that are not provided by Java standard library
- By inheriting from the standard exception classes
  - extends Exception: compiler checked exception
  - extends RuntimeException: compiler unchecked exception
- Constructor
  - Passing more specific exception messages

Example: User Defined Exception (1/3)

```java
import java.util.Scanner;

class InvalidInputException extends Exception {
    public InvalidInputException(String message) {
        super(message);
    }
}

public class ExceptionBasedInputLoop {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int number = 0;
        boolean validInput = false;
        String input = null;
```

Example: User Defined Exception (2/3)

```
        while (!validInput) {
            try {
                System.out.print("Please enter a positive odd integer: ");
                input = scanner.nextLine();
                number = Integer.parseInt(input); // convert String to int
                if (number <= 0) {  // negative integer
                    throw new InvalidInputException("Negative integer");
                }
                else if (number % 2 == 0) { // even number
                    throw new InvalidInputException("Not odd integer");
                }
                validInput = true; // exit from the loop if valid input
            } catch (InvalidInputException e) {
                System.out.println("Invalid input: " + e.getMessage());
            } catch (NumberFormatException e) {
                System.out.println("Invalid input: Not a valid integer");
            }
        }
```

Example: User Defined Exception (3/3)

```
        System.out.println("You entered a valid positive integer: " + number);
        scanner.close();
    }
}
```

```
Please enter a positive odd integer: a9832
Invalid input: Not a valid integer
Please enter a positive odd integer: -253
Invalid input: Negative integer
Please enter a positive odd integer: 2982
Invalid input: Not odd integer
Please enter a positive odd integer: 980751
You entered a valid positive integer: 980751
```