Object Oriented Programming
OOP 15 : Exception Handling

Hirra Anwar

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science
Islamabad

Agenda

- Introduction
- Difference between Error and Exception
- Dealing With Exceptions
- Throwing Exceptions
- Creating Your Own Exceptions

Agenda

- Introduction
- Difference between Error and Exception
- Dealing With Exceptions
- Throwing Exceptions
- Creating Your Own Exceptions

Introduction

- Exceptional event
- Error that occurs during runtime
- Cause normal program flow to be disrupted
- Examples
  - Divide by zero errors
  - Accessing the elements of an array beyond its range
  - Invalid input
  - Hard disk crash
  - Opening a non-existent file
  - Heap memory exhausted

Introduction

- **Benefits of handling exceptions:-**
- separating the error logic or exception logic from our regular business logic.
- grouping and differentiating the exception types.
- handling the exception and making the program to terminate normally or successfully.

Java Exception Hierarchy

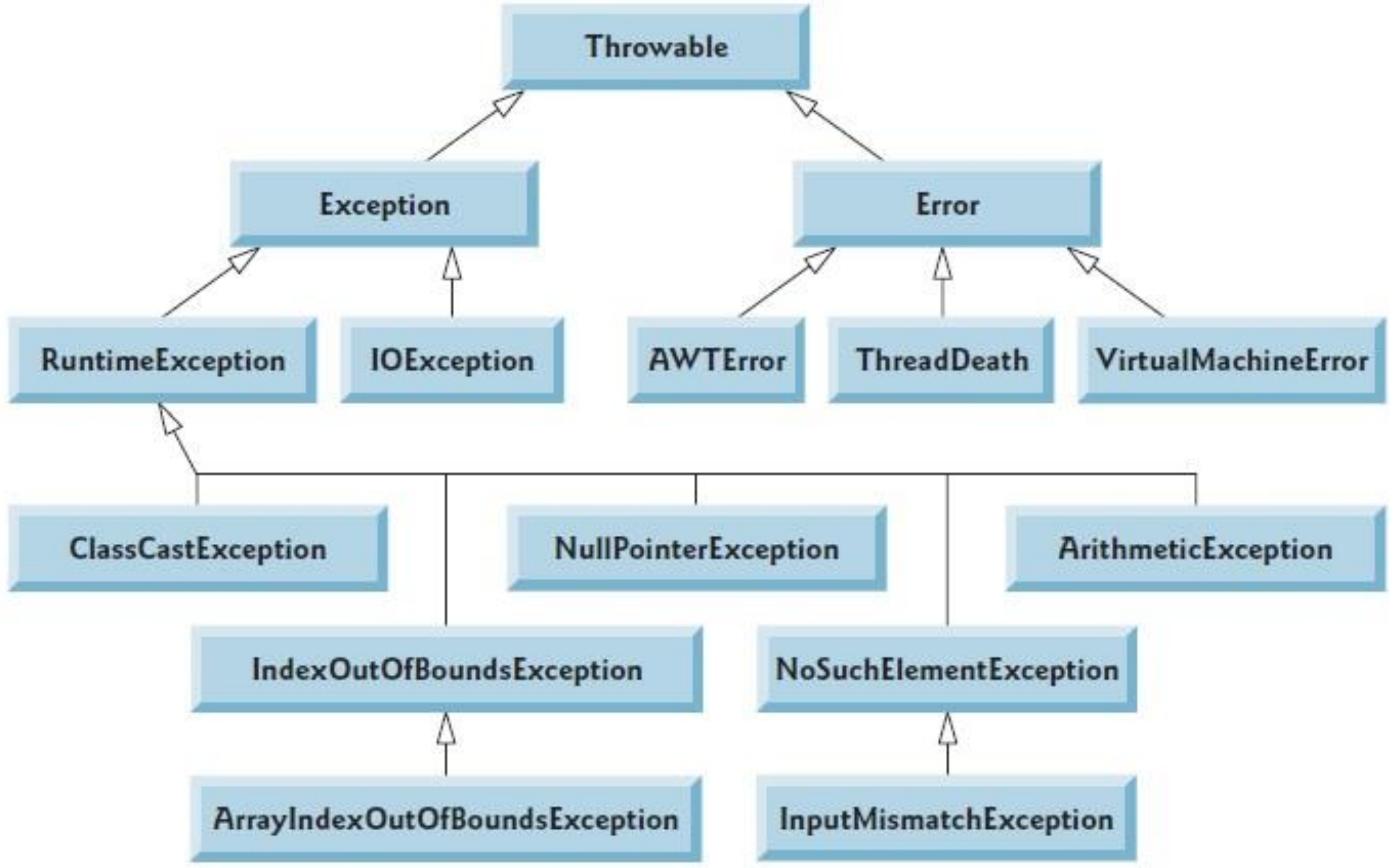


Fig. 11.3 | Portion of class `Throwable`'s inheritance hierarchy.

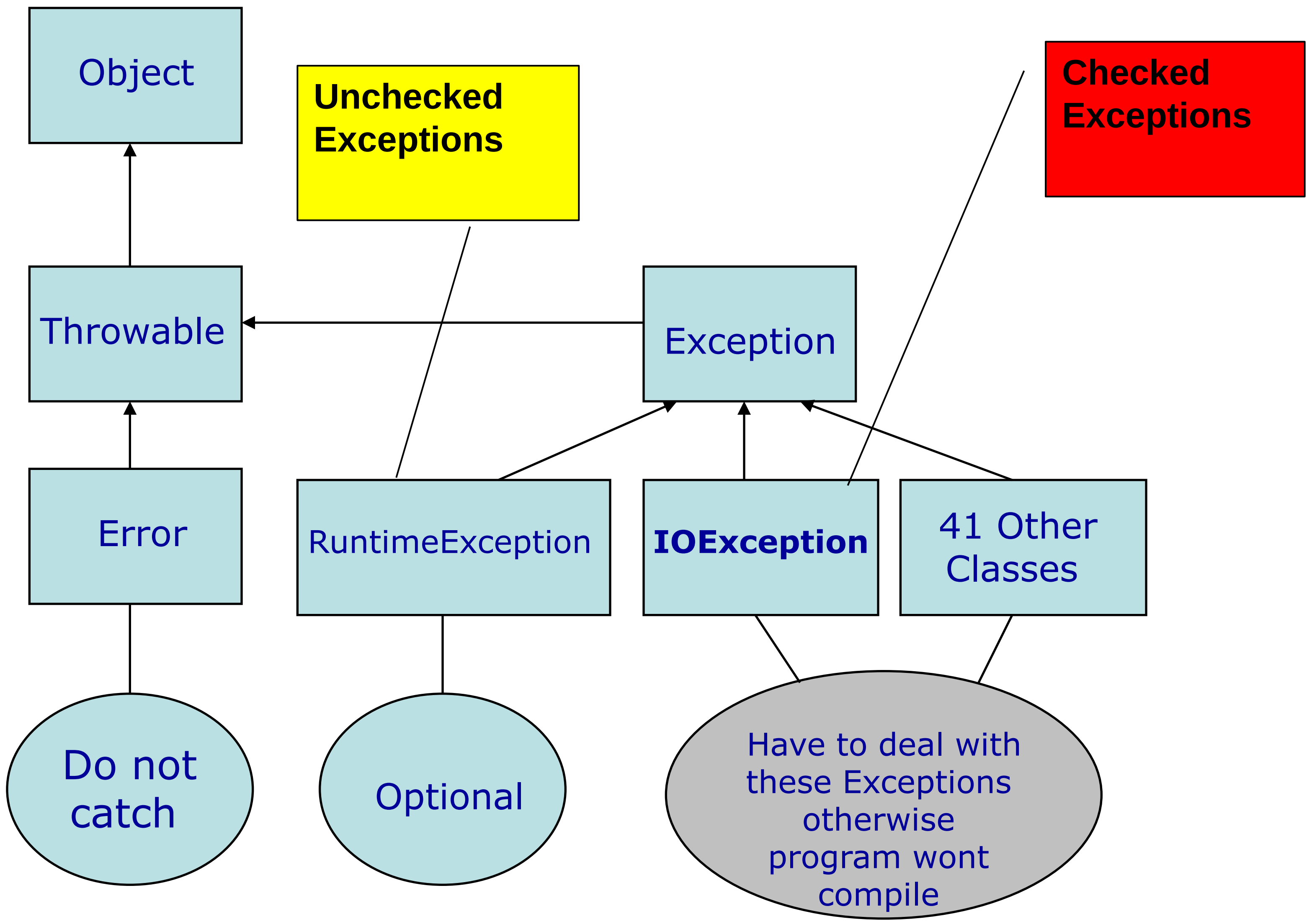
Introduction : Java Exceptions

- An exception in Java is an object that is created when an abnormal situation arises in your program
- This object has members that stores information about the nature of the problem
- An Exception is always an object of some subclass of the standard class Throwable
- Java provides a very well defined hierarchy of Exceptions to deal with situations which are unusual.
- All standard exceptions are covered by two direct subclasses of the class Throwable
  - Class **Error**
  - Class **Exception**

Exceptions 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.

Introduction : Java Exceptions



Agenda

- Introduction
- Difference between Error and Exception
- Dealing With Exceptions
- Throwing Exceptions
- Creating Your Own Exceptions

Difference between Error and Exceptions

- Error class
  - Used by the Java run-time system to handle errors occurring in the run-time environment
  - Generally beyond the control of user programs
  - Examples
    - Out of memory errors
    - Hard disk crash
- Exception class
  - Conditions that user programs can reasonably deal with
  - Usually the result of some flaws in the user program code
  - Examples
    - Division by zero error
    - Array out-of-bounds error

Exception Example

```
class DivByZero {
 public static void main(String args[]) {
     System.out.println(3/0);
     System.out.println(“Pls. print me.”);
}
```

- **Exception in thread "main“ java.lang.ArithmeticException: \ by zero at DivByZero.main(DivByZero.java:3)**
- Default exception handler
  - Provided by Java runtime
  - Prints out exception description
  - Prints the **stack trace**
  - Causes the program to terminate

  - A **Stack Trace** is produced automatically by the **Java**Virtual Machine when an **exception** is thrown to indicate the location and progression of the program up to the point of the **exception**.

Agenda

- Introduction
- Difference between Error and Exception
- Dealing With Exceptions
- Throwing Exceptions
- Creating Your Own Exceptions

Exceptions Handling

- When an exception occurs within a method, the method creates an exception object and hands it off to the runtime system
  - Creating an exception object and handing it to the runtime system is called "throwing an exception"
  - Exception object contains information about the error, including its type and the state of the program when the error occurred

```
/ File Name : ExcepTest.java
import java.io.*;

public class ExcepTest {

   public static void main(String args[]) {

         int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);

         System.out.println("Exception thrown  :" + e);
    }
```

Output:

Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3

```
/ 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);
      }
}
```

Output:

```
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3
```

Exceptions Handling

- **For all subclasses of Exception Class(except RuntimeException) you must include code to deal with them**
- **If your program has the potential to generate an exception of such a type, you have got two choices**
  - Handle the exception within the method
  - Register that your method may throw such an exception (You are passing the exception on)
- **If you do neither your code won't compile**

Exceptions Handling : try-catch

Syntax:

```
try
{
    <code to be monitored for exceptions>
}
catch (<ExceptionType1> <ObjName>)
{
    <handler if ExceptionType1 occurs>…
}
catch (<ExceptionTypeN> <ObjName>)
{
    <handler if ExceptionTypeN occurs>…
}
```

Exceptions Handling :Example-1

```
class DivByZero
{
   public static void main(String args[])
    {
        try
       {
         System.out.println(3/0);
         System.out.println(“Please print me.”);
       }
        catch (ArithmeticException exc)
       {
         //Division by zero is an ArithmeticException
         System.out.println(exc);
       }
       System.out.println(“After exception.”);
    }
}
```

Exceptions Handling :Example-2

```
class MultipleCatch
{
    public static void main(String args[])
    {
        try
        {
            int den = Integer.parseInt(args[0]);
            System.out.println(3/den);
        }
        catch (ArithmeticException exc)
        {
            System.out.println(“Divisor was 0.”);
        }
        catch (ArrayIndexOutOfBoundsException exc2)
        {
            System.out.println(“Missing argument.”);
        }
        System.out.println(“After exception.”);
    }
}
```