# Exceptions-Part 1

Chapter 7, Core Java Volume I

# Contents

- Dealing with Errors
- The Classification of Exceptions
- Runtime Exception Examples
- Catching an Exception
- Catching Multiple Exceptions
- Declaring Exceptions
- Catch or Declare Rule
- Throwing Exceptions
- Defining Exception Classes
- Rethrowing and Chaining Exceptions
- The `finally` Clause
- The Try-with-resources Statement
- Tips for Proper Use of Exceptions

# Dealing with Errors

- In a perfect world:
  - Users never enter invalid data
  - Files chosen by users to open always exist
  - The code has no logical error
- However, practically, our code should deal with the real world of bad data and buggy code.
- What kind of errors do you need to consider?
  - User input errors (e.g. malformed URL)
  - Device errors (e.g. printer truned off, network failed, etc.)
  - Physical limitations (e.g. run out of memory)
  - Code errors (e.g. invalid array index, pop empty stack, etc.)
  - etc.
- The programmer should:
  - notify the user by giving description about the error
  - save all work
  - allow the user to exit the program gracefully.

# Dealing with Errors

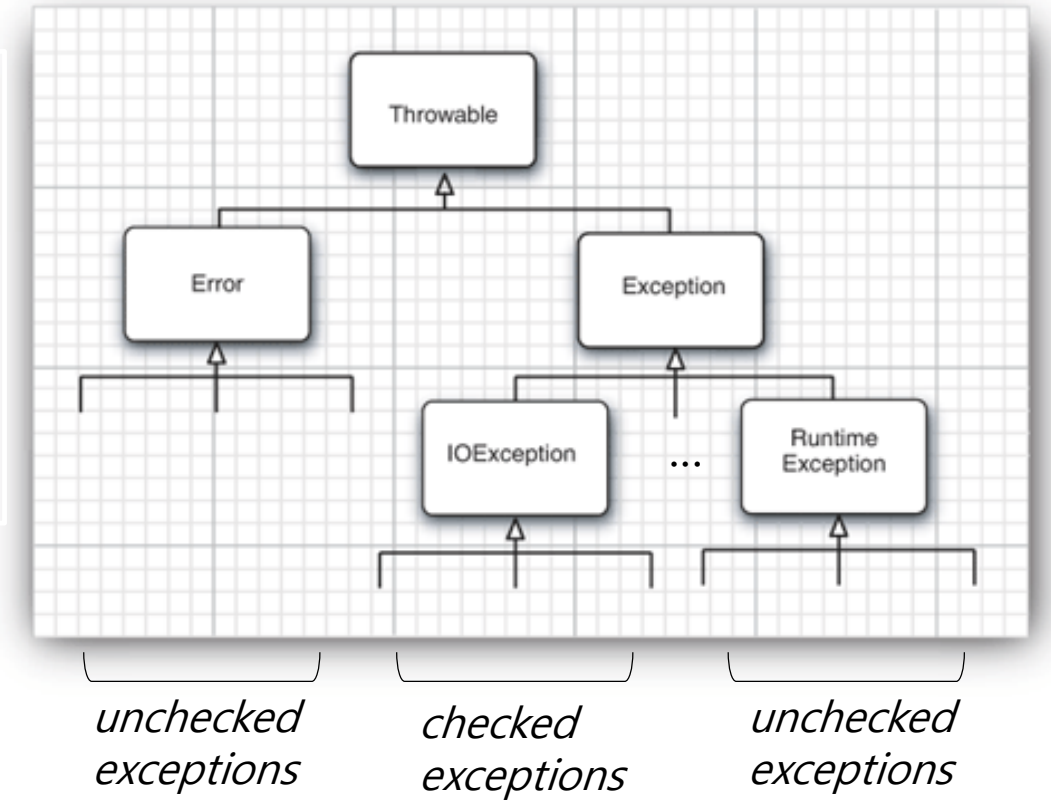
- Tranditional way to deal with errors
  - in a called method
    - Return an error code (-1, null, etc).
  - In a calling method
    - Check the error code.
    - Display error messages.
    - Terminate the program or handle the error and continue to execute

- Java provides an systematic way of *exception handling*
  - Mechanism for Throwing and Catching Exceptions
  - Predefined Exception Classes
  - User-defined Exception Classes

# The Classification of Exceptions

- In Java, we have  a hierarchy of predefined exception classes.

- Programmers can define their own exception classes inheriting the classes in the hierarchy.



# The Classification of Exceptions

- `Error` hierarchy:
  - Describes internal errors and resource exhaustion situations inside the Java runtime system.
  - Indicates serious problems that a reasonable applications shoud not try to catch.
  - Programmer should not throw an object of this type
- `Exception` hierarchy: a java programmer should  focus on this type.
- *Runtime Exceptions*: happen when a programmer make a mistake
  - Examples: bad cast, null pointer access, out-of-bound array access, etc.
  - In principle, instead of writing exception handling code for runtime exceptions, the programmer should write correct business logic.
- *IO Exceptions*: these are unavoidable facts of life such as
  - Trying to read past the end of a file.
  - Trying to open a file that doesn't exist.
  - Trying to find a `Class` object for a string that does not denote an existing class.

# The Classification of Exceptions

- Checked Exceptions
  - The exceptions that are checked at compile time.
  - If some code within a method throws a checked exception, then the method must either handle the exception or it must declare the exception using *throws* keyword.
  - All exceptions except **RuntimeException** and **Error** and their subclasses.
- Unchecked Exceptions
  - The exceptions that are not checked at compiled time.
  - **RuntimeExceptions** and **Error** and their subclasses.
  - It is up to the programmers to specify or catch the exceptions.

# Runtime Exception Examples

```
public class ExceptionTest
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Please enter numerator: ");
      int numerator = scanner.nextInt();
      System.out.print("Please enter denominator:
");
      int denominator = scanner.nextInt();

      int result = quotient(numerator, denominator);
      System.out.printf(
         "%nResult: %d / %d = %d%n", numerator,
denominator, result);
   }
```

```
   // demonstrates throwing a divide-by-zero exception
    public static int quotient(int num, int denom)
    {
       return num / denom; // possible division by zero
    }

} // end class ExceptionTest
```

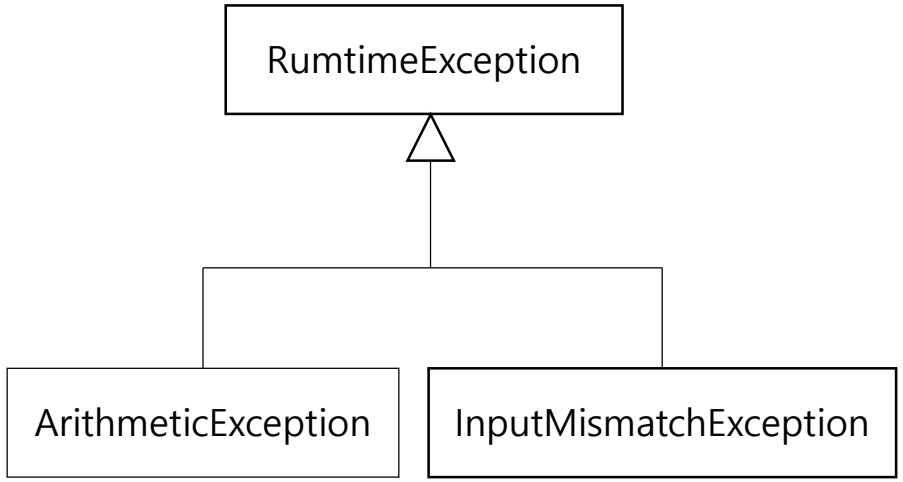
# Runtime Exception Examples

- Testing ArithmeticException

```
Please enter numerator: 10
Please enter denominator: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DivideByZeroTest.quotient(ExceptionTest.java:23)
        at DivideByZeroTest.main(ExceptionTest.java:16)
```

- Testing InputMismatchException

```
Please enter numerator: hello
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at DivideByZeroTest.main(ExceptionTest.java:12)
```



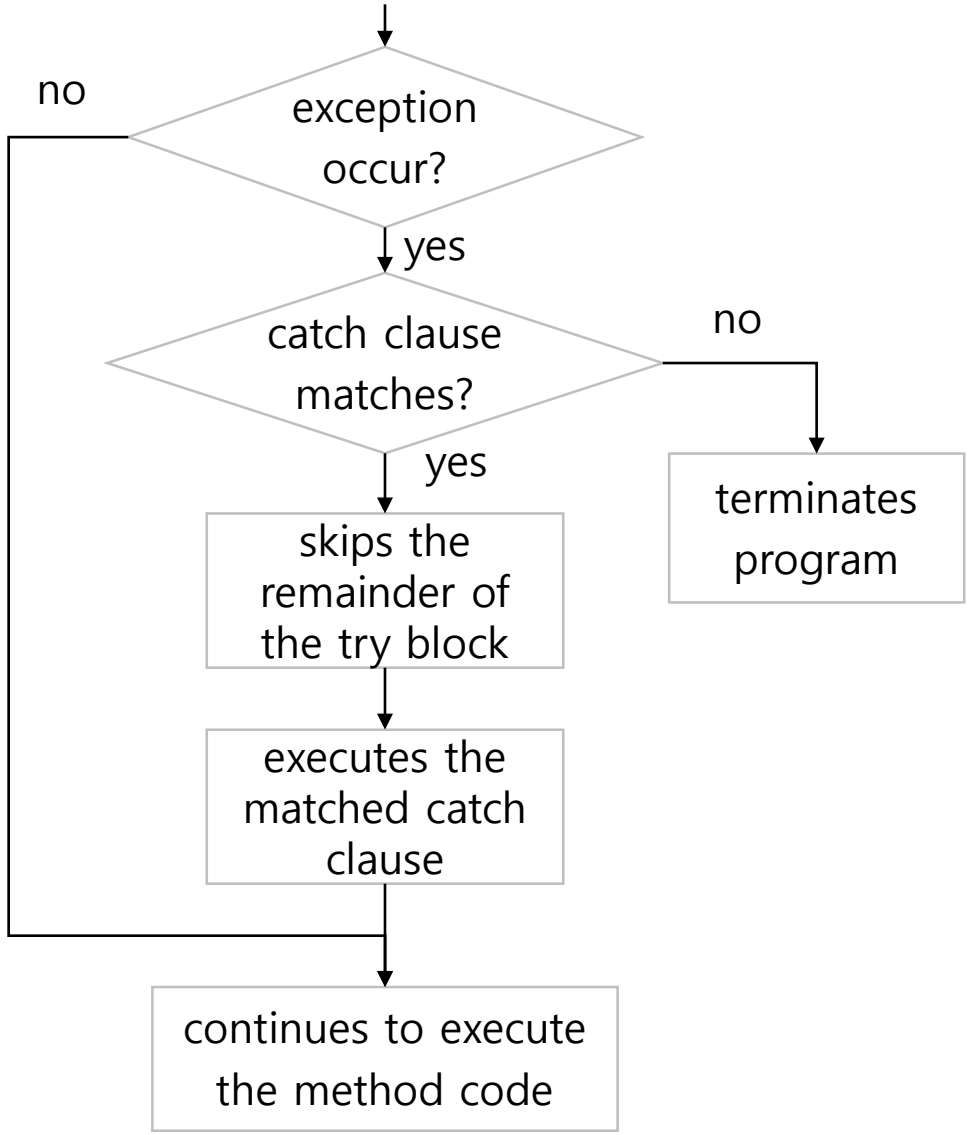
# Catching an Exception

- Use a try/catch block to catch an exception:

```
try
{
    code that might throw exceptions
}
catch (ExceptionType e)
{
    handler for the exceptions for this type
}

more catch clauses ...

method continues ...
```



# Catching an Exception

```
public class ExceptionHandlingTest
{
  public static void main(String[] args)
  {
    Scanner scanner = new Scanner(System.in);
    boolean continueLoop = true;
    do
    {
      try
      {
        System.out.print("Please enter  numerator: ");
        int numerator = scanner.nextInt();
        System.out.print("Please enter denominator: ");
        int denominator = scanner.nextInt();
        int result = quotient(numerator, denominator);
        System.out.printf("%nResult: %d / %d = %d%n",
numerator,denominator, result);
        continueLoop = false;
      }
      catch (InputMismatchException ex)
      {
        System.err.printf("%nException: %s%n", ex);
        scanner.nextLine();
        System.out.printf(
            "You must enter integers. Please try again.%n%n");
      }
    } while (continueLoop);
  }  // end of main

  public static int quotient(int num, int denom)
  {
    return num / denom; // not handled
  }
} // end class ExceptionHandlingTest
```

# Catching Multiple Exceptions

- You can catch multiple exceptions in separate catch clauses:

```
try
{
   code that might throw exceptions
}
catch (InputMismatchException e)
{
   emergency action for input mismatch
}
catch (ArithmeticException e)
{
   emergency action for arithmetic exception such as divide by zero
}
catch (RuntimeException e)
{
   emergency action for all other runtime exceptions
}
```

*Work with the inheritance hierarchy of exceptions:*
*Catch more specific exceptions before more general ones.*

➔ *If a catch clause with more general exception locates above those with more specific ones, compiler issues error.*

# Declaring Exceptions

- If you write a method that might throw an exception, you can declare that fact with a throws clause:

```
public static int quotient(int num, int denom) throws ArithmeticException { … }
```

- A throws clause can list multiple exceptions:

```
public int nextInt() InputMismatchException, NoSuchElementException { … }
```

- You can find a lot of exception declarations in library classes

```
public Scanner(File source) throws FileNotFoundException  // in Scanner class
public FileInputStream(String name) throws FileNotFoundException // in FileInputStream class
public int read() throws IOException  // in FileInputStream class
```

- You don't have to declare unchecked exceptions (i.e. runtime exceptions).
- But, you have to declare checked exceptions unless the exceptions are catched.

# Declaring Exceptions

```
public class ExceptionHandlingTest
{
  public static void main(String[] args)
  {
    Scanner scanner = new Scanner(System.in);
    boolean continueLoop = true;
    do
    {
      try
      {
        System.out.print("Please enter  numerator: ");
        int numerator = scanner.nextInt();
        System.out.print("Please enter denominator: ");
        int denominator = scanner.nextInt();
        int result = quotient(numerator, denominator);
        System.out.printf("%nResult: %d / %d = %d%n",
numerator,denominator, result);
        continueLoop = false;
      }
      catch (InputMismatchException ex)
      {
        System.err.printf("%nException: %s%n", ex);
        scanner.nextLine();
        System.out.printf("You must enter integers.%n%n");
      }
      catch (ArithmeticException ex)
      {
        System.err.printf("%nException: %s%n", ex);
        System.out.printf("Zero is an invalid denominator.%n%n");
      }
    } while (continueLoop);
  }  // end of main
  public static int quotient(int num, int denom)
        throws ArithmeticExamples // you don't have to declare it
  {
    return num / denom; // not handled
  }
} // end class ExceptionHandlingTest
```