Exception Handing in Java

Program defects and ‘bugs’

- During my demo in the class or your working on your assignment, do you often encounter programming errors?

Program defects and ‘bugs’

- Defects or problems often appear in software after it is delivered
  - It can be difficult to test a software product completely in the environment in which it is used
    - Because …?
  - How to prevent this?
    - Understand the defects
    - Catch them before they occur, and then?
    - Handle them, if they do happen!
      - Avoid surprise or un-expectation!

Major categories of defects

- Syntax and semantic errors
- Run-time errors and exceptions
- Logic Errors

Major categories of defects

- Syntax and semantic errors
  - grammatical mistakes or violation of rules specified in a programming language
  - The compiler detects syntax errors, that must be corrected to compile successfully
  - Some common errors include:
    - Omitting or misplacing braces, parentheses, etc.
    - Performing an incorrect operation on a primitive type value
    - Invoking an instance method not defined
    - Not declaring a variable before using it
    - Providing multiple declarations of a variable
    - Failure to import a library routine

Major categories of defects

- Live demo:
  - Syntax and semantic errors

Major categories of defects

- Run-time errors and exceptions
  - Occur during program execution (run-time!)
  - Occur when the JVM detects an operation that it knows to be incorrect
  - Cause the JVM to throw an exception
  - Examples of run-time errors include
    - Division by zero
    - Array index out of bounds
    - Number format error
    - Null pointer exceptions

Major categories of defects

- Live demo:
  - Runtime errors

Major categories of defects

- Examples of run-time errors:

```
//ArithmeticException
int a=50/0;

//NullPointerException
String s=null;
System.out.println(s.length());

//NumberFormatException
String s="abc";
int i=Integer.parseInt(s);

//ArrayIndexOutOfBoundsException
int a[]=new int[5];
a[10]=50;
```

Major categories of defects

- Difference between divided by 0 and 0.0?

```
// ArithmeticException is thrown!
System.out.println("50/0 = " + (50/0));

// = Infinity
System.out.println("50/0.0 = " + (50/0.0));

// = Infinity
System.out.println("50.0/0.0 = " + (50/0.0));

// = NaN (not a number)
System.out.println("0/0.0 = " + (0/0.0));
```

Major categories of defects

- Difference between divided by 0 and 0.0?

```
// ArithmeticException is thrown!
System.out.println("50/0 = " + (50/0));

// = Infinity
System.out.println("50/0.0 = " + (50/0.0));
… …
```

  - Java follows the standard IEEE-754, for floating point math, which mandates division by zero to return a special "infinity" value.

    Throwing an exception would actually violate that standard.
  - Java 'Double' class: https://docs.oracle.com/javase/7/docs/api/java/lang/Double.html
  - More discussions here: https://stackoverflow.com/questions/12954193/why-does-division-by-zero-with-floating-point-or-double-precision-numbers-not

Major categories of defects

- Logic error – programmer's mistake in
  - the *design* of a class or method, or
  - the *implementation* of an algorithm
  - *is difficult to find* - usually non-detectable by the compiler, and not cause run-time errors
    - The code runs perfectly as written — it just isn't performing the task that you expected it to perform.
    - How to find them then?
      - through testing
      - through users (not a good idea?)

Major categories of defects

- Live demo
  - logical errors

Major categories of defects

- Examples of logic errors:

```
// Using incorrect operator precedence
// compute values according to the formula F = G.m1.m2 /
r2
double force = G * mass1 * mass2 / r * r;

// Relying on integer values to measure values
int r=2;
int area = 2*2*PI

// misplacement of semicolon
int count = 20;
for(int i=0; i<count; i++);
     System.out.println("Count is " + i);
```

Major categories of defects

- Logic error – how to avoid?
  - Consider “corner” / extreme cases
  - Have reviews / walk-throughs: other eyes
    - Pair programming
  - Use library/published algorithms where possible

Major categories of defects

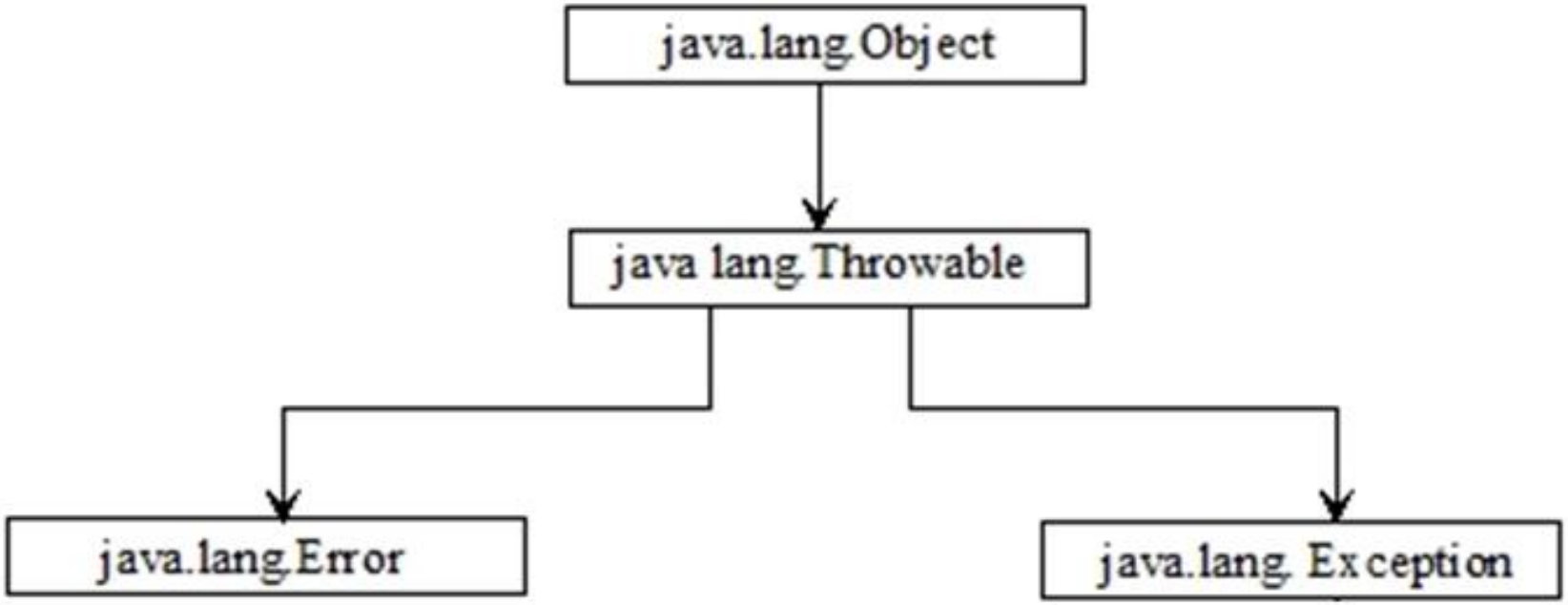
- Now we have seen exceptions in Java may cause a program to terminate immature.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at HelloWorld.main(HelloWorld.java:11)
```

- How to handle the exceptions in Java?

Exception handling in Java

- The *exception* is a class in Java, inherited from the class *Throwable*:



Exception handling in Java

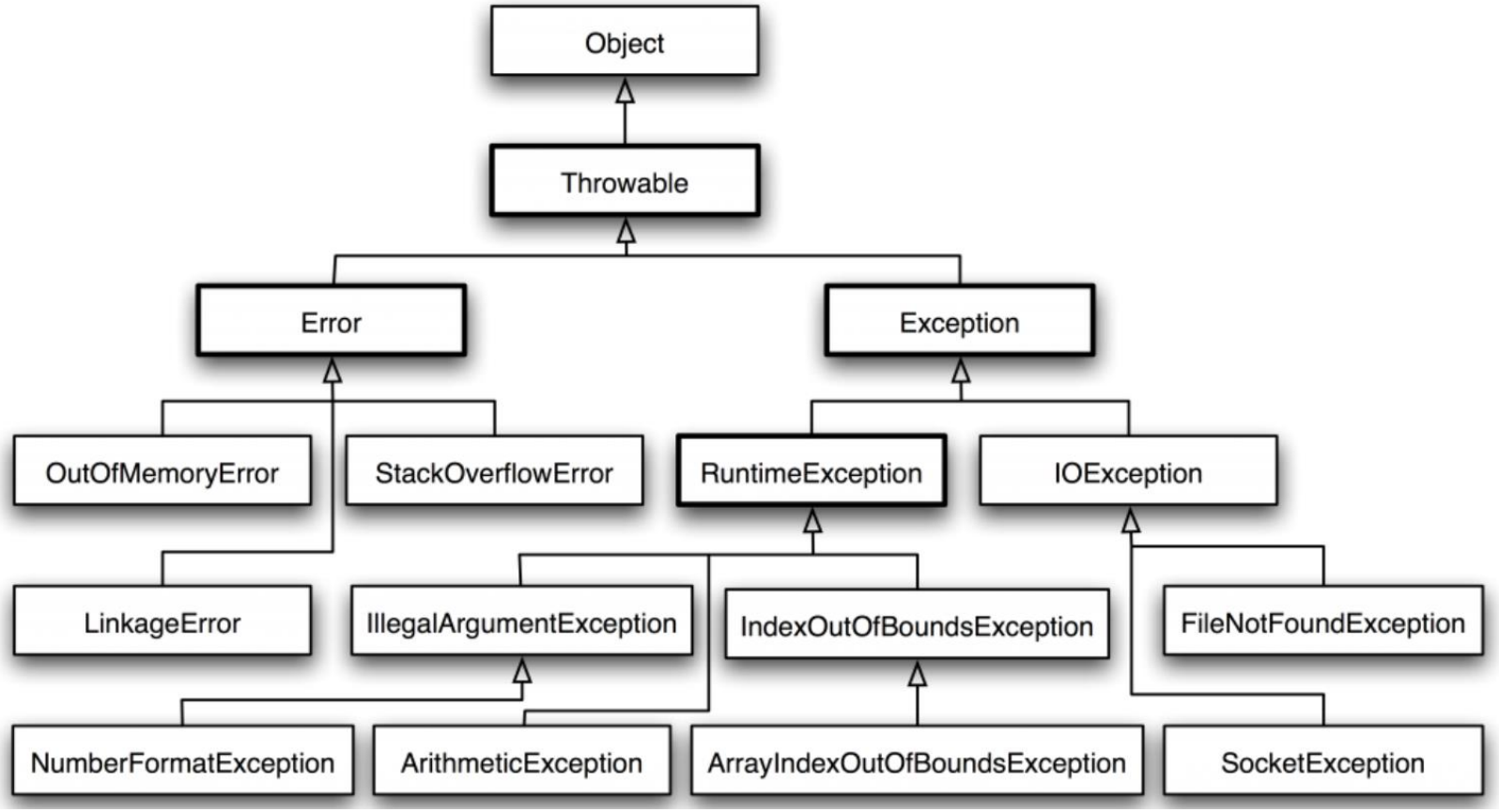
- Let's have a close look at the 'Throwable' class
  - https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Throwable.html

Summary of Commonly Used Methods from the `java.lang.Throwable` Class

Method	Behavior
`String getMessage()`	Return the detail message.
`void printStackTrace()`	Print the stack trace to `System.err`.
`String toString()`	Return the name of the exception followed by the detail message.

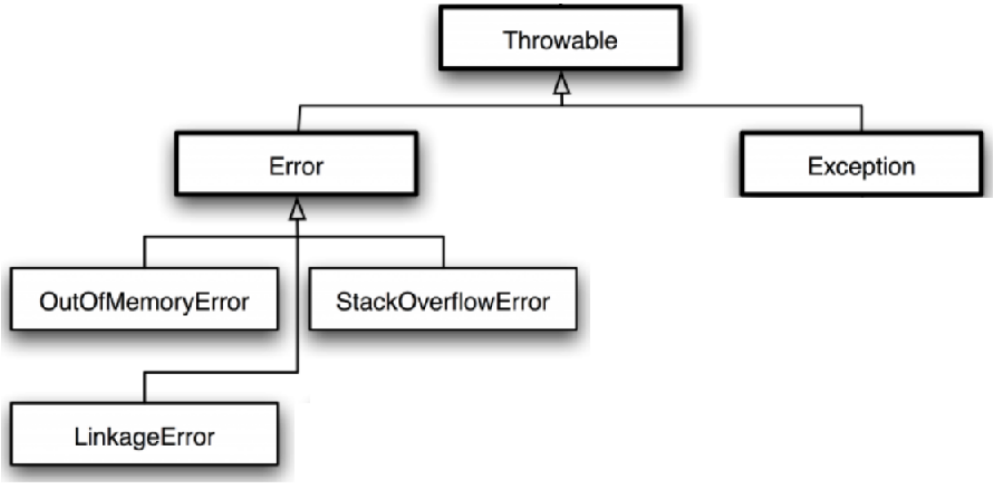
Exception handling in Java

- Two subclasses, ‘Error’ and ‘Exception’, under Throwable:



Exception handling in Java

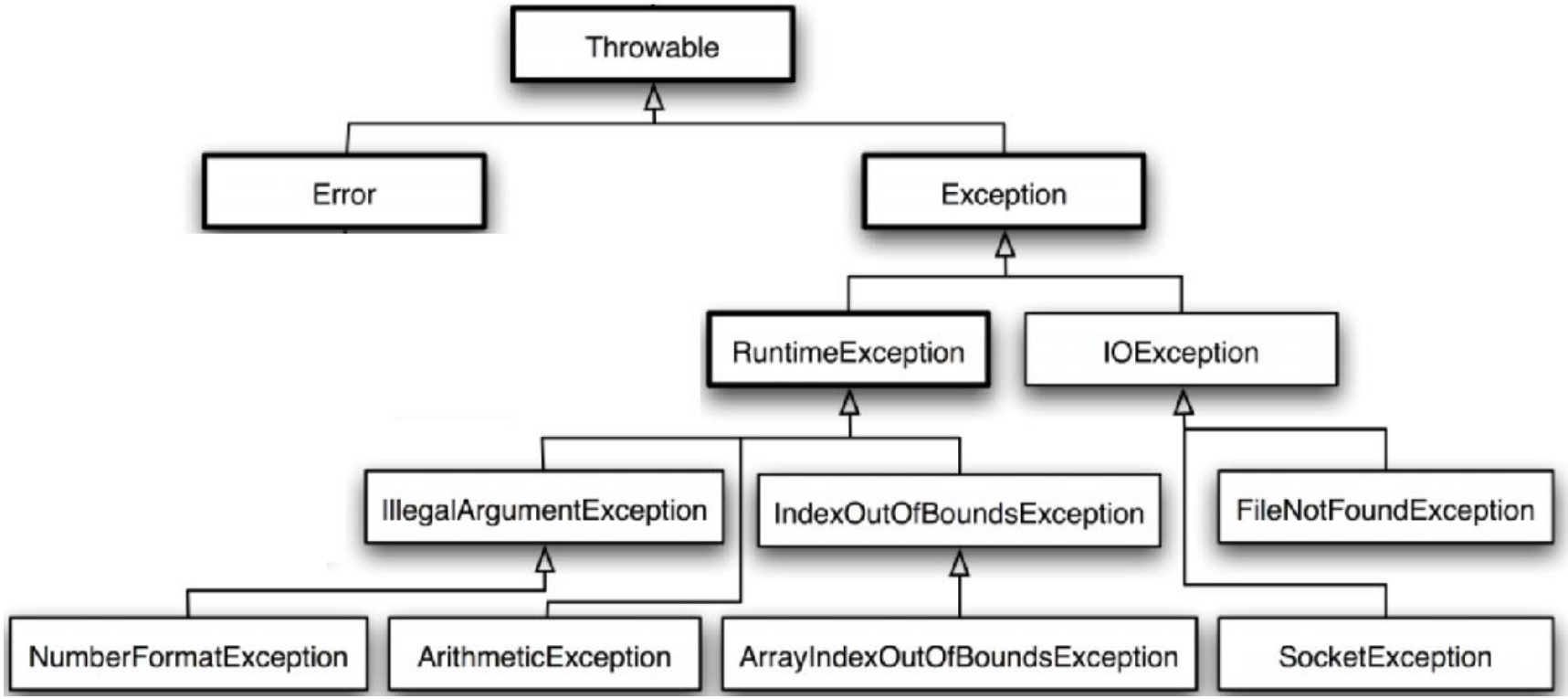
- The subclass, ‘Error’:



- OutOfMemeoryError: thrown when an attempt to allocate memory fails
- StackOverflowError: thrown when a stack overflow error occurs within the virtual machine
- LinkageError: thrown when there is a problem resolving a reference to a class. Reasons for this may include a difficulty in finding the definition of the class

Exception handling in Java

- The subclass, 'Exception':



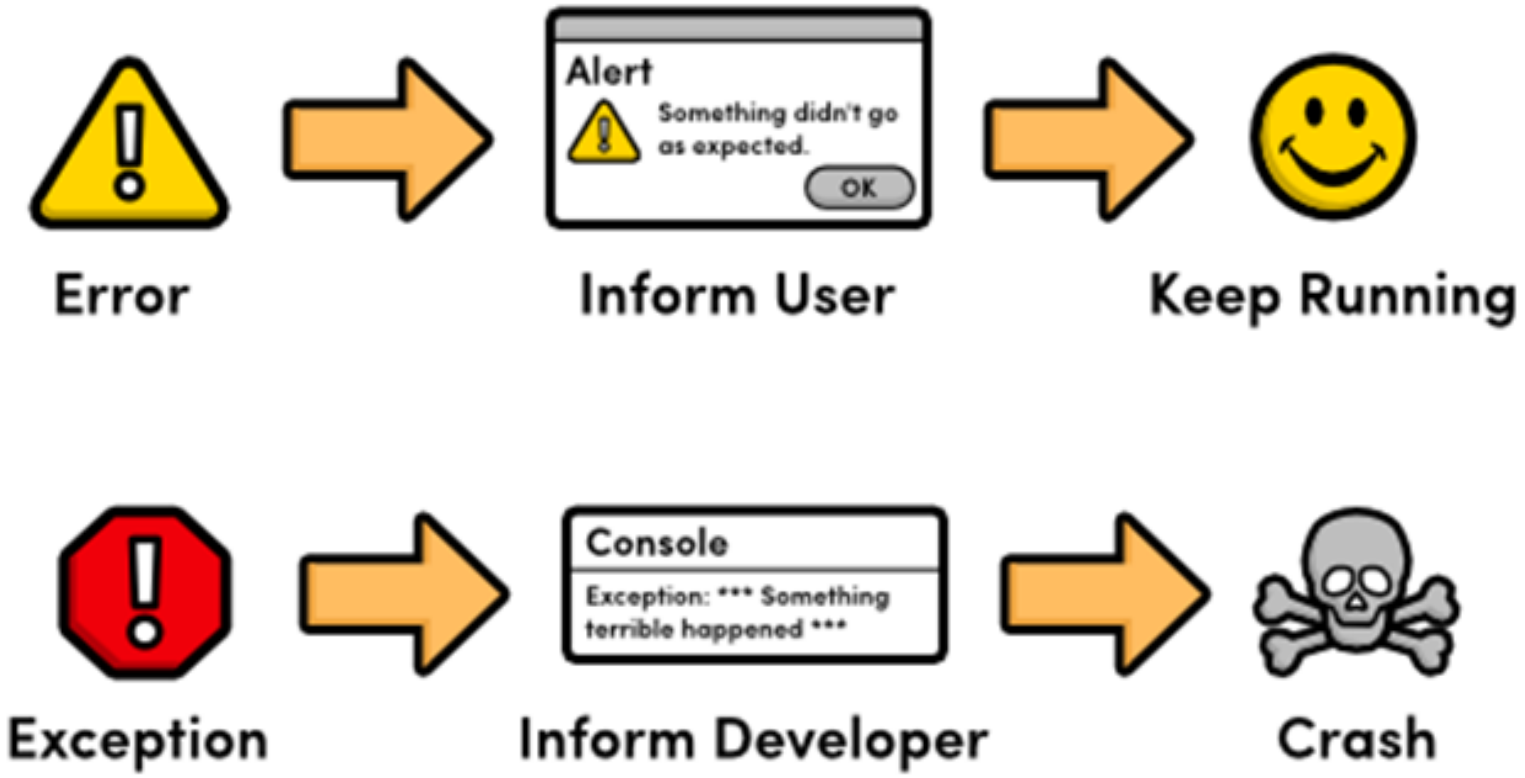
Exception handling in Java

- Then, the question is, “what are the differences between an Error and an exception”?
  - E.g., ‘OutOfMemoryError’ vs ‘IndexOutOfBoundsException’?

  - Error:
    - due to lack of system resources
    - out of programming scope as such type of error can’t predicted
    - non-recoverable
  - Exception:
    - an unexpected and unwanted event that disturbs normal flow of the program
    - often due to programmatic logic
    - recoverable

Exception handling in Java

- Due to the differences between Error and Exception, when they occur, we take different actions:



Exception handling in Java

- For errors - serious problems, a reasonable application should not try to catch or handle.
  - An error (problem) is outside the application's scope of handling
  - The application cannot anticipate that it will happen as it may occur randomly, so it cannot recover from it
  - For instance, these kind of errors may involve poor internet connection on the part of the user or maybe they are experiencing hardware failure.
    - In the middle of execution of request to the server, the internet connection is abruptly cut

Exception handling in Java

- For exceptions: it is highly recommended to handle them, and the main objective of exception handling is graceful termination of the program

  - For instance, our program is to read data from a remote file locating at a server at runtime. If the remote file is not available, we will get an exception saying *FileNotFoundException*.

    - If *FileNotFoundException* occurs, we can tell the program to try a local file and continue the rest of the program normally.

    - So, really two steps here? Catch the exception and handle it?

Exception handling in Java

- Let's see
  - how your Java program may throw exceptions
  - how you as a programmer may catch them, and then handle them properly

```
public static void main(String args[])
{
  try{
       //code that may raise exception
       int result = dividedBy(int userInput);
  }
  catch(ArithmeticException e)
  {
       System.out.println(e);
  }

  //rest code of the program
  System.out.println("rest of the code...");
}
```

Exception handling in Java

- Handle single exception
  - Place the statements that can cause an exception inside a ***try*** block, and the handler inside a ***catch*** clause

```
public static void main(String args[])
{
   try{
        //code that may raise exception
        int result = dividedBy(int userInput);
   }
   catch(ArithmeticException e)
   {
        System.out.println(e);
   }

   //rest code of the program
   System.out.println("rest of the code...");
}
```

Exception handling in Java

- Handle multiple exceptions
  - Three exceptions may be thrown in the try block, they are …

```
try
{
    String filename = "myNote.txt";
    Scanner in = new Scanner(new File(filename));
    String input = in.next();
    int value = Integer.parseInt(input);
}

… …
```

- The Scanner constructor can throw a FileNotFoundException.
- Scanner.next can throw a NoSuchElementException.
- Integer.parseInt can throw a NumberFormatException.

Exception handling in Java

- Handle multiple exceptions
  - Three exceptions may be thrown in the try block:
    - The Scanner constructor can throw a FileNotFoundException.
    - Scanner.next can throw a NoSuchElementException.
    - Integer.parseInt can throw a NumberFormatException.

```
try{
    String filename = "myNote.txt";
    Scanner in = new Scanner(new File(filename));
    String input = in.next();
    int value = Integer.parseInt(input);
}
catch (FileNotFoundException e)
{
        e.printStackTrace();
}
catch (NoSuchElementException e)
{
        System.out.println(e.getMessage());
}
Catch (NumberFormatException e)
{
        // handle it here
}
```

- Question:
  - When someone throws a stone to you, what shall you do?
  - When someone blames you for a fault, what shall you do? What options may you have?

- Throw an exception

Exception handling in Java

- Throw an exception
  - Example one:

```
// add a new telephone number or changes an old one

public String addOrChangeEntry(String name, String number)
{
  if (!isPhoneNumberFormat(number))
  {
    throw new IllegalArgumentException(
                    "Invalid phone number: " + number);
  }
  ...
}
```

Exception handling in Java

- Throw an exception
  - Example two:

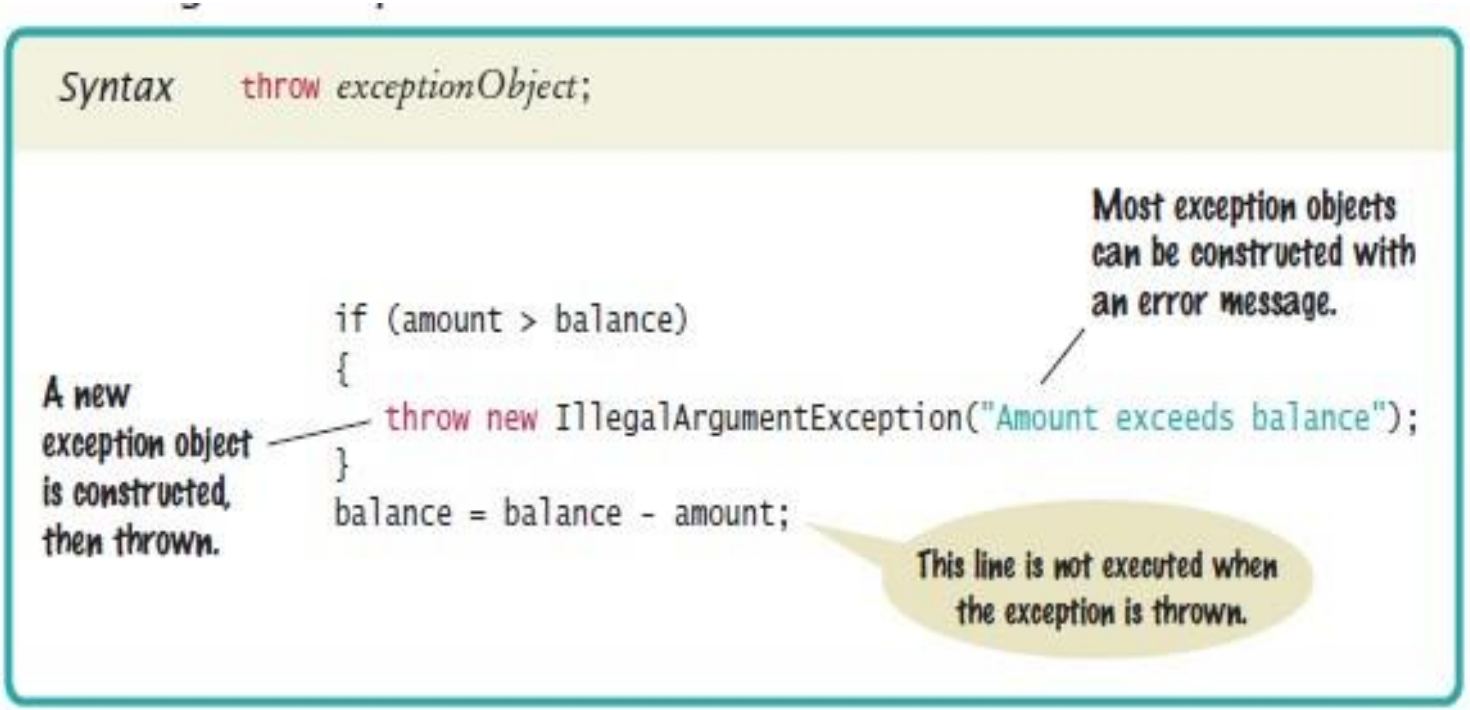
```
public void accessLocalFile (String askingUser)
    throws CertificateException
{
  ...

  if (user's secure socket certificate bad)
  {
    throw new CertificateException(reason);
  }
  ...
}
```

Exception handling in Java

- When an exception is thrown from a method, what will happen next?
  - The method terminates immediately!! Just like a circuit breaker that cuts off the flow of electricity in a dangerous situation.

Exception handling in Java

- Throw an exception
  - Note that an exception is an instance of the class '*Exception*'



- Customized exceptions

Exception handling in Java

- Design your own exceptions
  - Also known as custom exception or user-defined exception
  - These are the exceptions related to business logic and workflow. It is useful for the application users or the developers to understand the exact problem.
  - Do so by '*extends*' Java built-in class 'Exception':

```
public class MyDefinedException extends Exception
{
    public MyDefinedException(String errorMessage)
    {
       super(errorMessage);  // ??
    }
}
```

Exception handling in Java

- Design your own exceptions
  - Here is an example

```
class InvalidAgeException  extends Exception
{
    public InvalidAgeException (String str)
    {
        // calling the constructor of parent Exception
        super(str);
    }
}
```

Exception handling in Java

- Design your own exceptions
  - Use the customized exception

```
public class TestCustomException1
{
    // method to check the age
    static void validate (int age) throws InvalidAgeException
    {
      if(age < 18)
      {

        // throw an object of user defined exception
        throw new InvalidAgeException("Invalid age to vote");
      }
      else
      {
          System.out.println("welcome to vote");
      }
    }

    public static void main(String[] args)
    { … … }
}
```

Exception handling in Java

- Design your own exceptions
  - Use the customized exception

```
public class TestCustomException1
{
    static void validate (int age) throws InvalidAgeException
    {
       if(age < 18) throw new InvalidAgeException( … )
    }

    public static void main(String[] args)
    {
       try {
            validate(13);
       }
       catch (InvalidAgeException e)
       {
           System.out.println("Exception occured: " + e);
       }
     }
}
```

- When an exception occurs, as a programmer/developer, you may give a final word!
  - Offered a chance to redeem yourself -☺

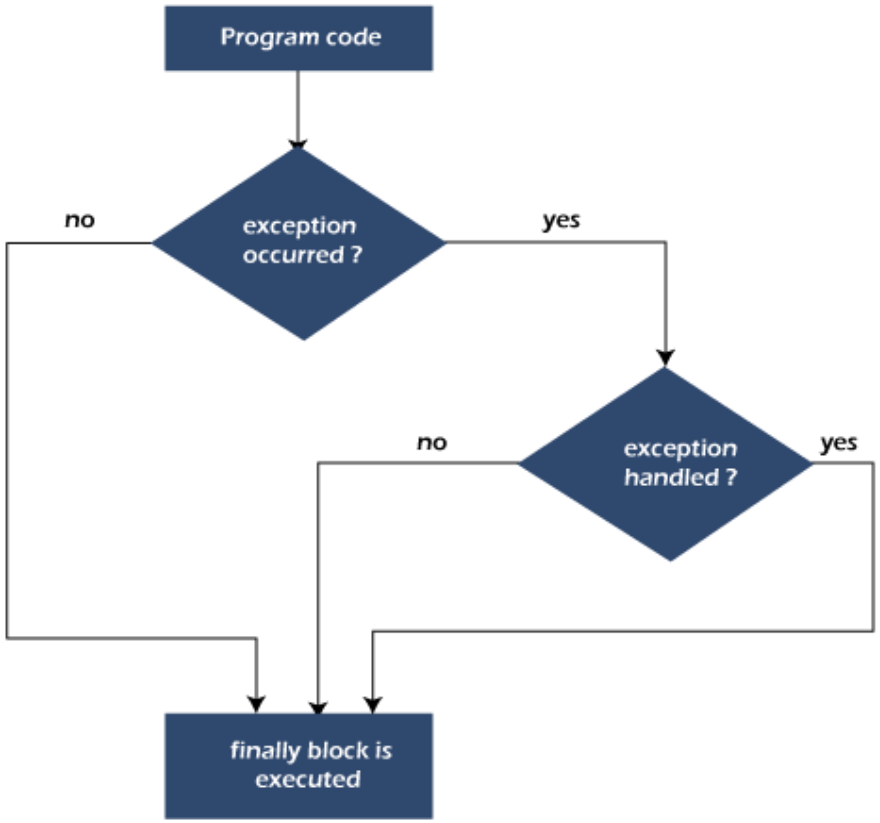
Exception handling in Java

- try – finally statement
  - A block is always executed whether an exception is *handled* or not.

```
try
{
        . . .
}
catch (…)
{
        …
}
finally
{
        … …
  // This code is executed whether or not an exception occurs

}
```

Exception handling in Java

- try – finally statement
  - it contains all the necessary statements that need to be printed regardless of the exception *occurs* or not.
  - If you don't handle the exception, what is going to happen with before terminating the program, JVM executes finally block (if any).



Exception handling in Java

- try – finally statement
  - Why using the ‘finally’ block?
  - to "clean up" code such as closing a file, closing connection, etc.
  - The important statements (messages) to be printed can be placed in the finally block.

Exception handling in Java

- try – finally statement
  - What's output of the codes below?

```
public static void main(String args[])
{
  try {
       int data=25/5;
       System.out.println(data);
  }
  catch(NullPointerException e)
  {
       System.out.println("exception is handled");
  }
  finally
  {
       System.out.println("in the finally block …");
  }

  System.out.println("rest of the code...");
}
```

Exception handling in Java

- try – finally statement
  - What's output of the codes below?

```
public static void main(String args[])
{
   try {
        int data=25/0;
        System.out.println(data);
   }
   catch(NullPointerException e)
   {
        System.out.println("exception is handled");
   }
   finally
   {
        System.out.println("in the finally block …");
   }

   System.out.println("rest of the code...");
}
```

Exception handling in Java

- try – finally statement
  - What's output of the codes below?

```
public static void main(String args[])
{
  try {
       int data=25/0;
       System.out.println(data);
  }
  catch(ArithmeticException e)
  {
       System.out.println("exception is handled");
  }
  finally
  {
       System.out.println("in the finally block …");
  }

  System.out.println("rest of the code...");
}
```