Object-Oriented Programming

Exceptions

Outline

- Concept of exception
- Throwing and catching exceptions
- Rethrowing exceptions
- Tracing exceptions

What is Exception?

- Exception is an indication of problem that arises during the execution of a program
- Exception happens in case of:
  - Designing errors
  - Programming errors
  - Data errors
  - System errors
  - ...

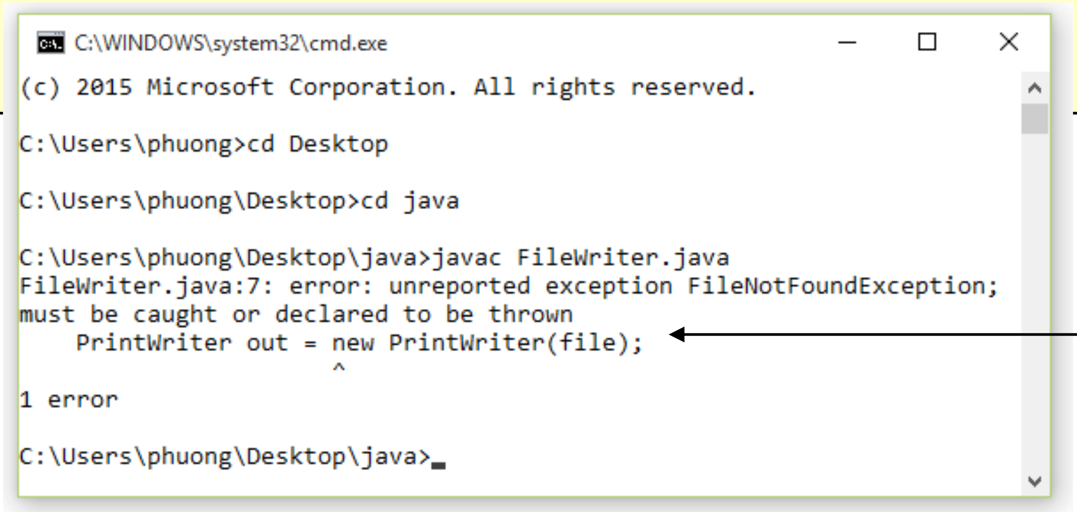
Example: Open File

```
import java.io.PrintWriter;
import java.io.File;

class FileWriter {
  public static void write(String fileName, String s)  {
    File file = new File(fileName);
    PrintWriter out = new PrintWriter(file);

    out.println(s);
    out.close();
  }
}
```

Open file to write



```
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\phuong>cd Desktop

C:\Users\phuong\Desktop>cd java

C:\Users\phuong\Desktop\java>javac FileWriter.java
FileWriter.java:7: error: unreported exception FileNotFoundException;
must be caught or declared to be thrown
    PrintWriter out = new PrintWriter(file);
                      ^
1 error

C:\Users\phuong\Desktop\java>_
```

Compile-time error

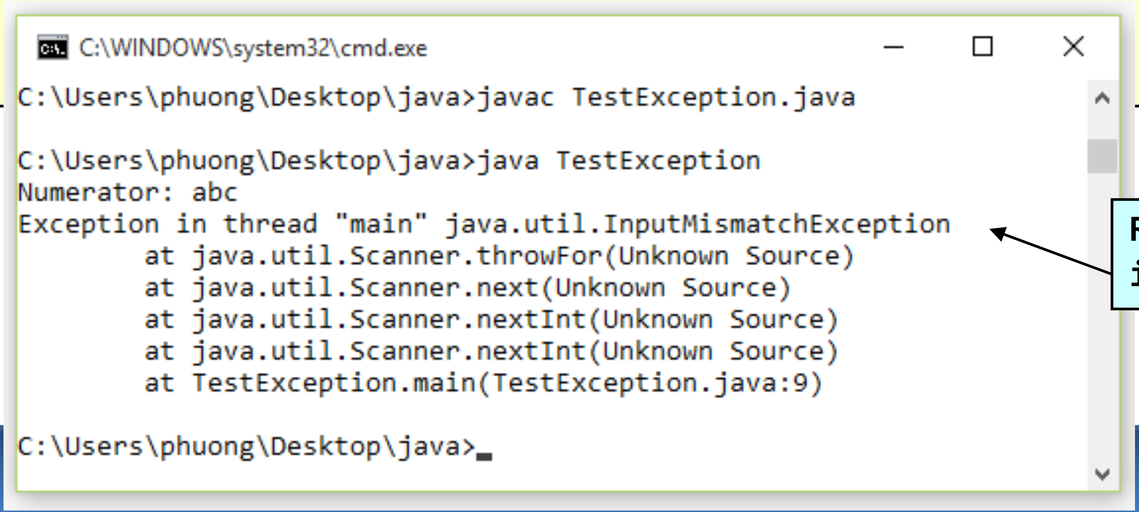
Example: Invalid Input

```
import java.util.*;
public class TestException
{
   public static void main (String args[]) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Numerator: ");
      int numerator = scanner.nextInt();
      System.out.print("Denominator: ");
      int denominator = scanner.nextInt();

      int result = numerator/denominator;

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

What happens if input is not a valid integer?


C:\WINDOWS\system32\cmd.exe

```
C:\Users\phuong\Desktop\java>javac TestException.java

C:\Users\phuong\Desktop\java>java TestException
Numerator: abc
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 TestException.main(TestException.java:9)

C:\Users\phuong\Desktop\java>_
```

Runtime error by invalid integer input “abc”

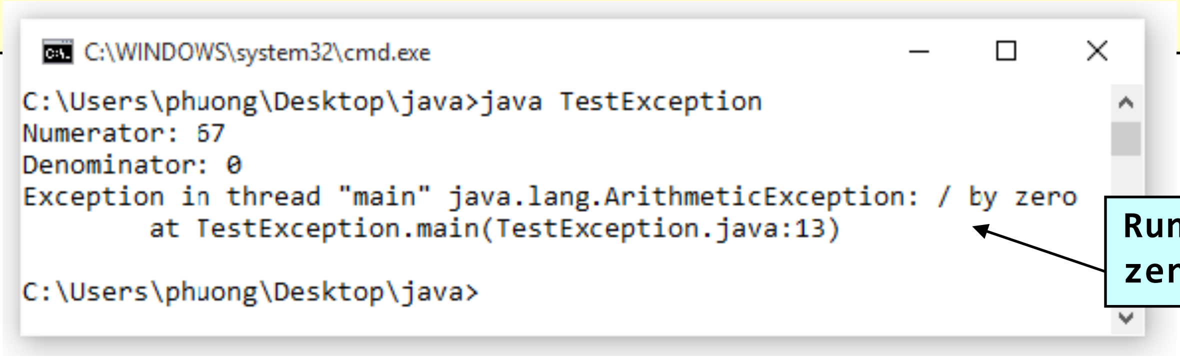
Example: Divide by Zero

```
import java.util.*;
public class TestException
{
   public static void main (String args[]) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Numerator: ");
      int numerator = scanner.nextInt();
      System.out.print("Denominator: ");
      int denominator = scanner.nextInt();

      int result = numerator/denominator;

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

What happens if denominator is zero?



```
C:\Users\phuong\Desktop\java>java TestException
Numerator: 67
Denominator: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at TestException.main(TestException.java:13)

C:\Users\phuong\Desktop\java>
```

Runtime error by dividing zero

Throwing exceptions

- Exception is thrown to an object that contains information about the error
- `throws` clause – specifies types of exceptions a method may throw
- Thrown exceptions can be:
  - in method’s body, or
  - from method’s header

Throwing exceptions

```
class Fraction {
    private int numerator, denominator;

    public Fraction (int n, int d) throws ArithmeticException
    {
        if (d==0)
            throw new ArithmeticException();
        numerator = n; denominator = d;
    }
}

public class TestException2 {
    public static void main(String [] args) {
        Fraction f = new Fraction (2,0);
    }
}
```

Declare what type of exceptions the method might throw

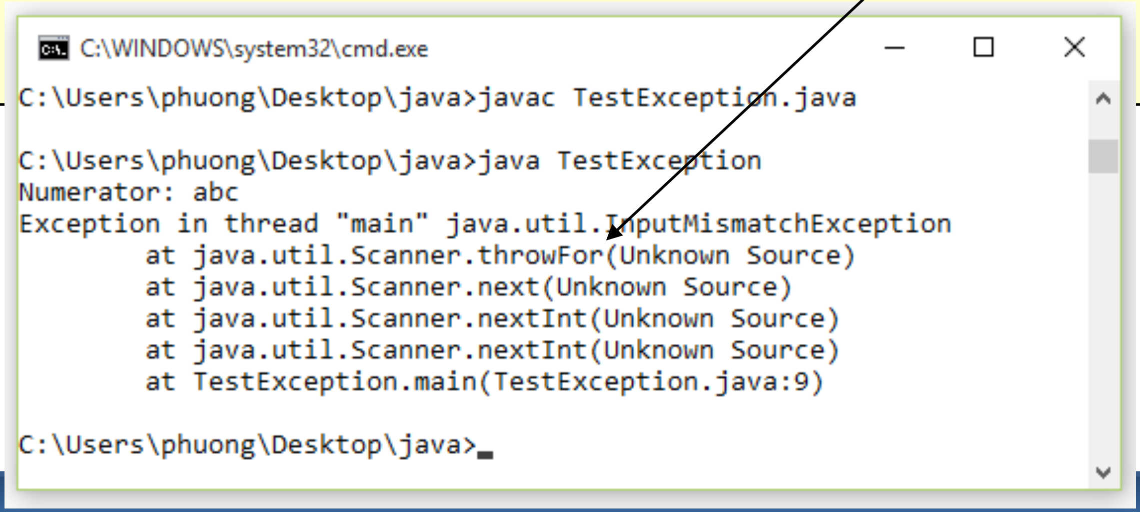
An ArithmeticException object is created and thrown in method's body

Throw Point

Throw point is the initial point at which the exception occurs

```
import java.util.*;
public class TestException
{
   public static void main (String args[]) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Numerator: ");
      int numerator = scanner.nextInt();
      …
   }
}
```

Throw Point



```
C:\Users\phuong\Desktop\java>javac TestException.java

C:\Users\phuong\Desktop\java>java TestException
Numerator: abc
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 TestException.main(TestException.java:9)

C:\Users\phuong\Desktop\java>
```

Catching exceptions

- Syntax:

```
try {
   // throw an exception
}
catch (TypeOfException e) {
   // exception-handling statements
}
```

- Separate the code that describes what you want to do (program logic) from the code that is executed when things go wrong (error handling)
  - try block – program logic: encloses code that might throw an exception and the code that should not be executed if an exception occurs
  - catch block – error handling: catches and handles an exception

Catching exceptions

- A **`catch`** block can catch:
  - Exception of the declared type:

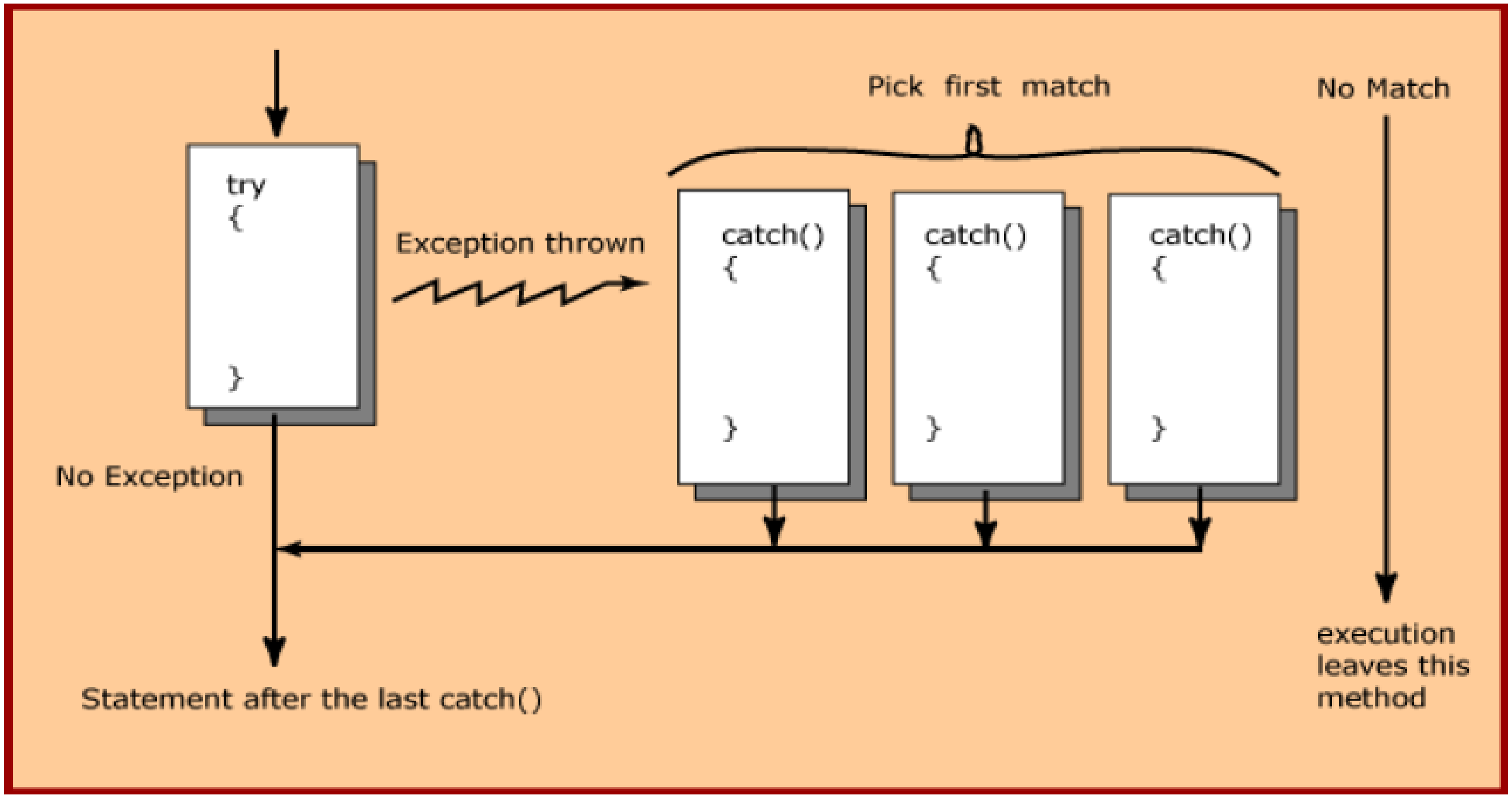
```
catch (IOException e) {
  // catch exceptions of type IOException
}
```

  - Exception of a subclass of the declared type:

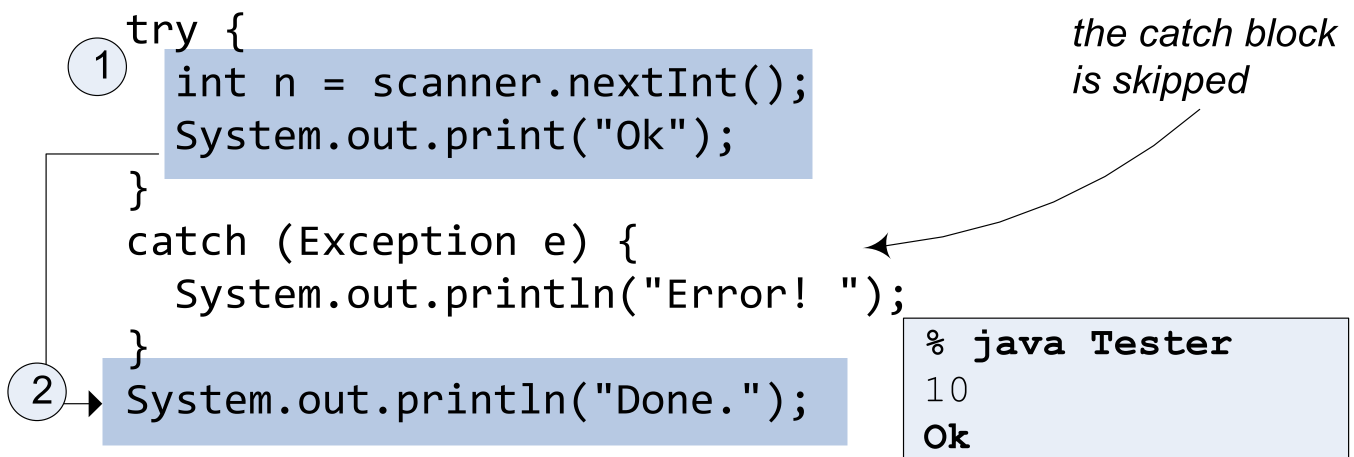
```
catch (IOException e) {
  // catch exceptions of type FileNotFoundException
  // or EOFException…
}
```

- Uncaught exception: an exception that occurs when there is no **`catch`** blocks matches

How **try** and **catch** work?



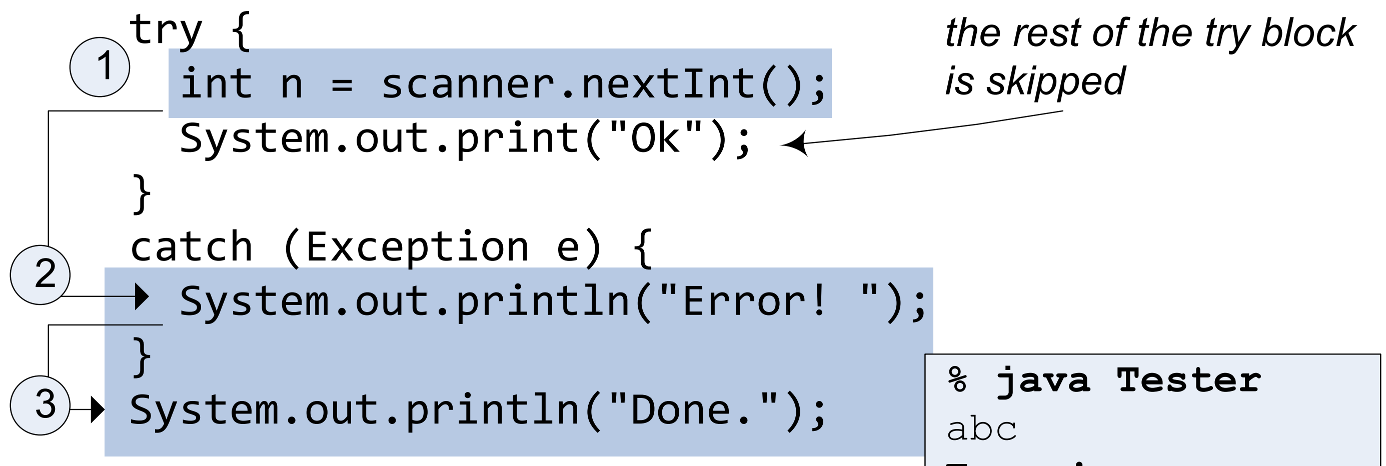
1. No errors



```
try {
  int n = scanner.nextInt();
  System.out.print("Ok");
}
catch (Exception e) {
  System.out.println("Error! ");
}
System.out.println("Done.");
```

```
% java Tester
10
Ok
Done.
```

2. The error is caught and handled



```
try {
  int n = scanner.nextInt();
  System.out.print("Ok");
}
catch (Exception e) {
  System.out.println("Error! ");
}
System.out.println("Done.");
```

```
% java Tester
abc
Error!
Done.
```

3. The error cannot be caught

```
  try {
1   int n = scanner.nextInt();
    System.out.print("Ok");
  }
  catch (ArithmeticException e) {
    System.out.println("Error! ");
  }
  System.out.println("Done.");
} // end of method
```

the rest of the method, is skipped

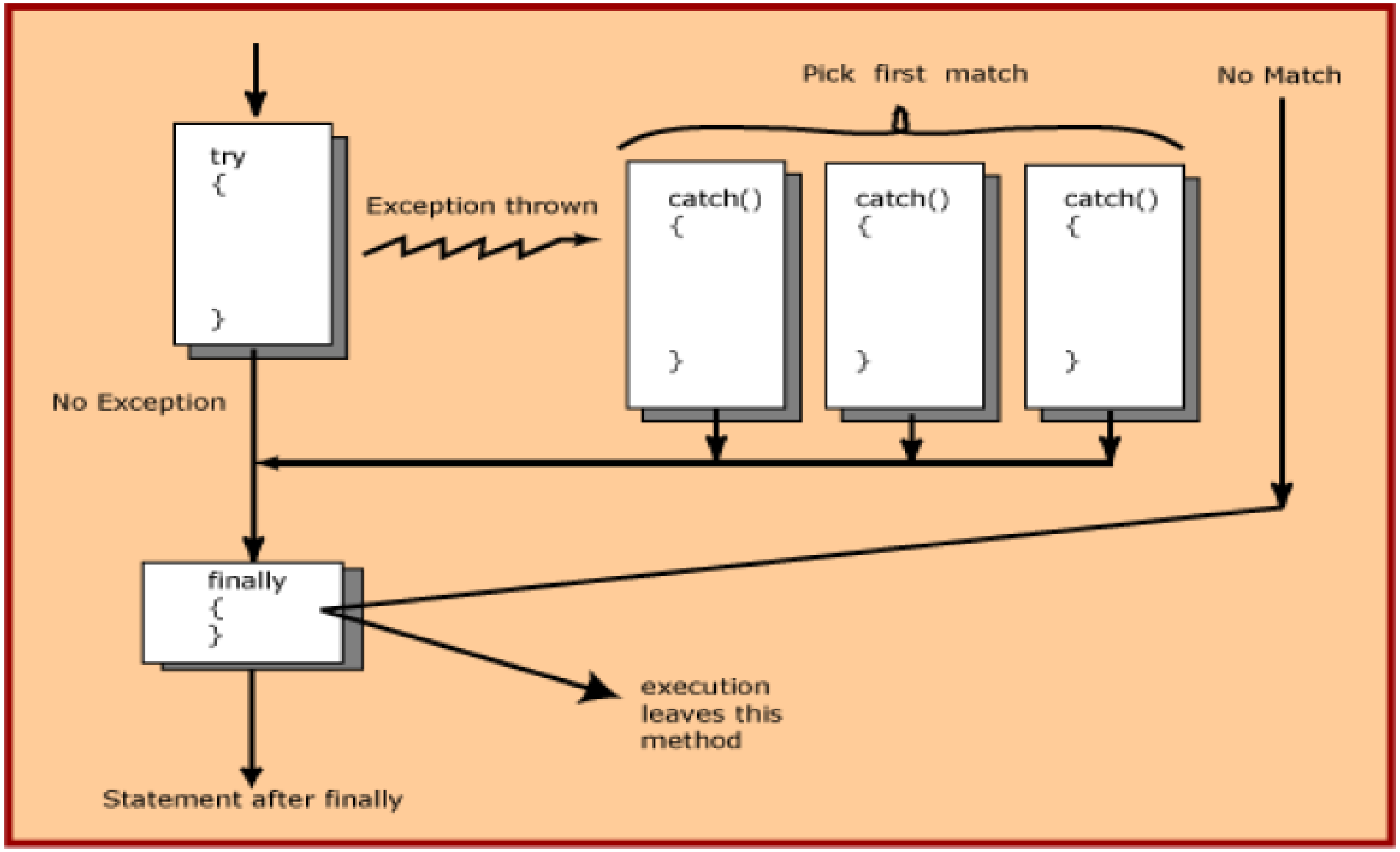
control gets out of the method

finally block

- Optional in a try statement
- Placed after last catch block
- Always executed, except when application exits from try block by method “System.exit()”
- Often contains resource-release code, such as file closing

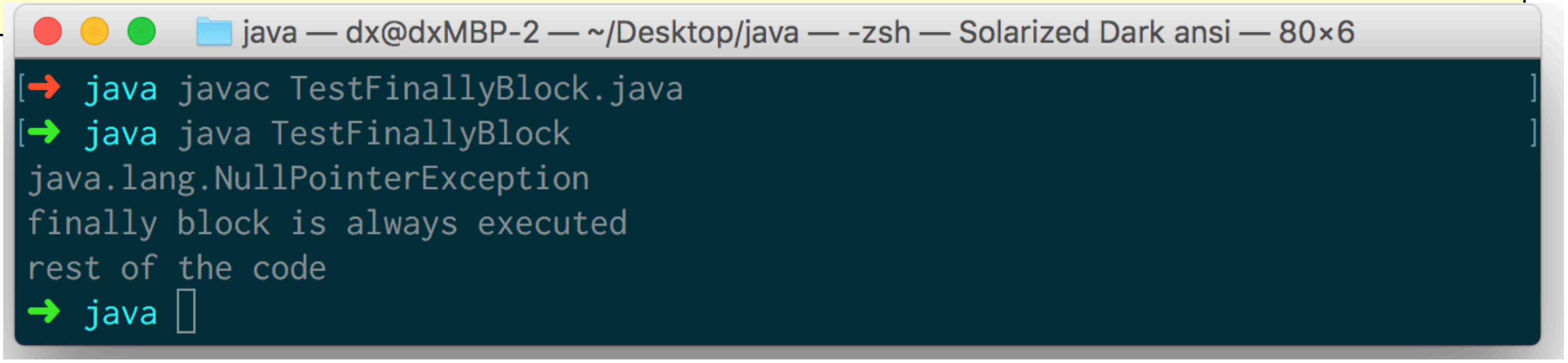
```
try {
…
}
catch(Exception1 e1) {
…
}
catch(Exception2 e2) {
…
}
finally {
…
}
```

How `finally` works?



Example: finally block

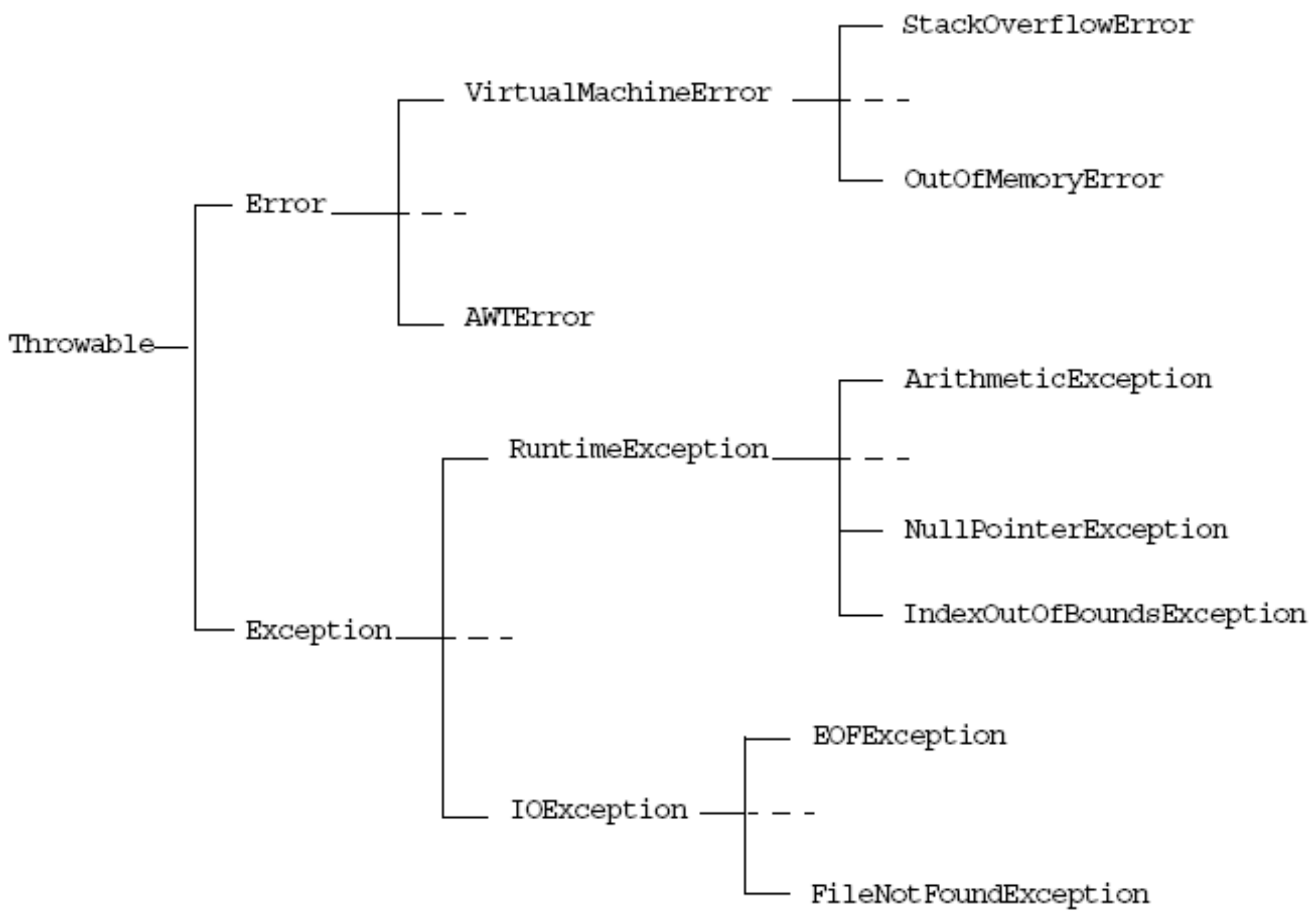
```java
public class TestFinallyBlock {
    public static void main(String args[]) {
        try {
            String a = null;
            System.out.println("a is " + a.toLowerCase());
        } catch (NullPointerException e) {
            System.out.println(e);
        } finally {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code");
    }
}
```



java — dx@dxMBP-2 — ~/Desktop/java — -zsh — Solarized Dark ansi — 80×6

```
➜ java javac TestFinallyBlock.java
➜ java java TestFinallyBlock
java.lang.NullPointerException
finally block is always executed
rest of the code
➜ java
```

Java Exception Hierarchy



Handling exceptions

- The goal is to resolve exceptions so that the program can continue or terminate gracefully
- Handling exception enables programmers to create programs that are more robust and fault-tolerant

Exception handling methods

Three choices to put to a method:

- catch and handle
  - try and catch blocks
- pass it on to the method’s caller
  - thrown exceptions
- catch, handle, then pass it on
  - re-thrown exceptions

Rethrowing exceptions

- Exceptions can be re-thrown when a `catch` block decides that:
  - it cannot process the exception, or
  - it can process the exception only partially

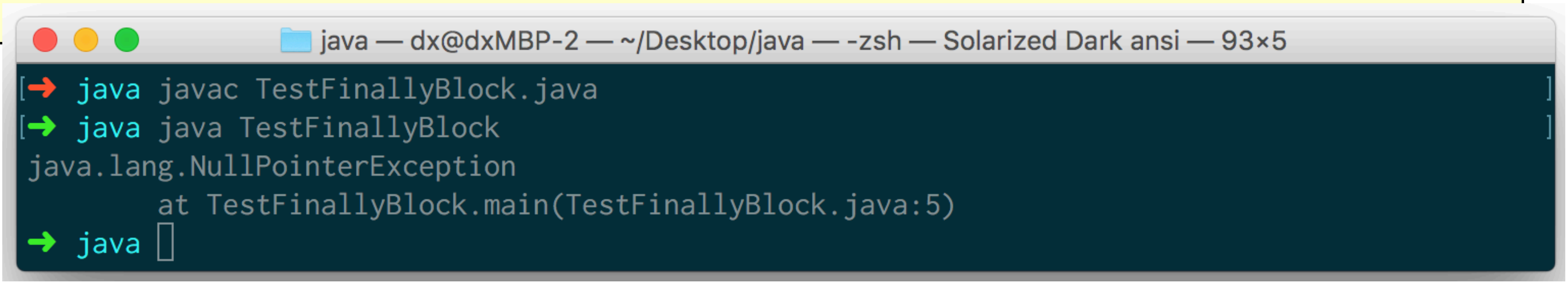
- Example:

```
try {...
}
catch (Exception e) {
    System.out.println(e.getMessage());
    throw e;
}
```

Tracing exceptions

- Can use **`printStackTrace()`** to trace back to the point where an exception was issued

```
public class TestFinallyBlock {
    public static void main(String args[]) {
        try {
            String a = null;
            System.out.println("a is " + a.toLowerCase());
        } catch (NullPointerException e) {
            e.printStackTrace();
        }
    }
}
```



java — dx@dxMBP-2 — ~/Desktop/java — -zsh — Solarized Dark ansi — 93×5

```
➜ java javac TestFinallyBlock.java
➜ java java TestFinallyBlock
java.lang.NullPointerException
        at TestFinallyBlock.main(TestFinallyBlock.java:5)
➜ java
```