Java Programming I

CHAPTER 09

Exceptions
Part 1

Contents

- Traditional Error Handling
- Exceptions Vs. Traditional
- What is an Exception?
- Kinds of Exceptions
- Exception Handling Keywords
- Catching and Handling Exceptions
- Keyword: finally

Traditional Error Handling

- Consider the following pseudo code for reading a file into memory:
  - A file is a collection of data (information) stored together on an external media such as hard disk, USB flash memory, SD card, etc. under a particular name.

```
readFile {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
}
```

Problems with readFile

- What happens if the file can't be opened?
- What happens if the length of the file can't be determined?
- What happens if enough memory can't be allocated?
- What happens if the read fails?
- What happens if the file can't be closed?

Error Handling with Traditional Programming

```
errorCodeType readFile {
    initialize errorCode = 0;
    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;
        } else {
            errorCode = errorCode and -4;
        }
    } else {
        errorCode = -5;
    }
    return errorCode;
}
```

- Here text in red is regular code
- Text in black is exception handling code

Error Handling in Java

```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

- Here text in red is regular code
- Text in black is exception handling code

- Note that the error handling code and ``regular'' code are separate!

Traditional Error Handling

- It is difficult to separate error-handling code from regular code!
- How can we provide some systematic way for the error handling in Java? → By using Exceptions

Exceptions Vs. Traditional

Q: What are the advantages/disadvantages of Exception handling?

Exceptions Vs. Traditional

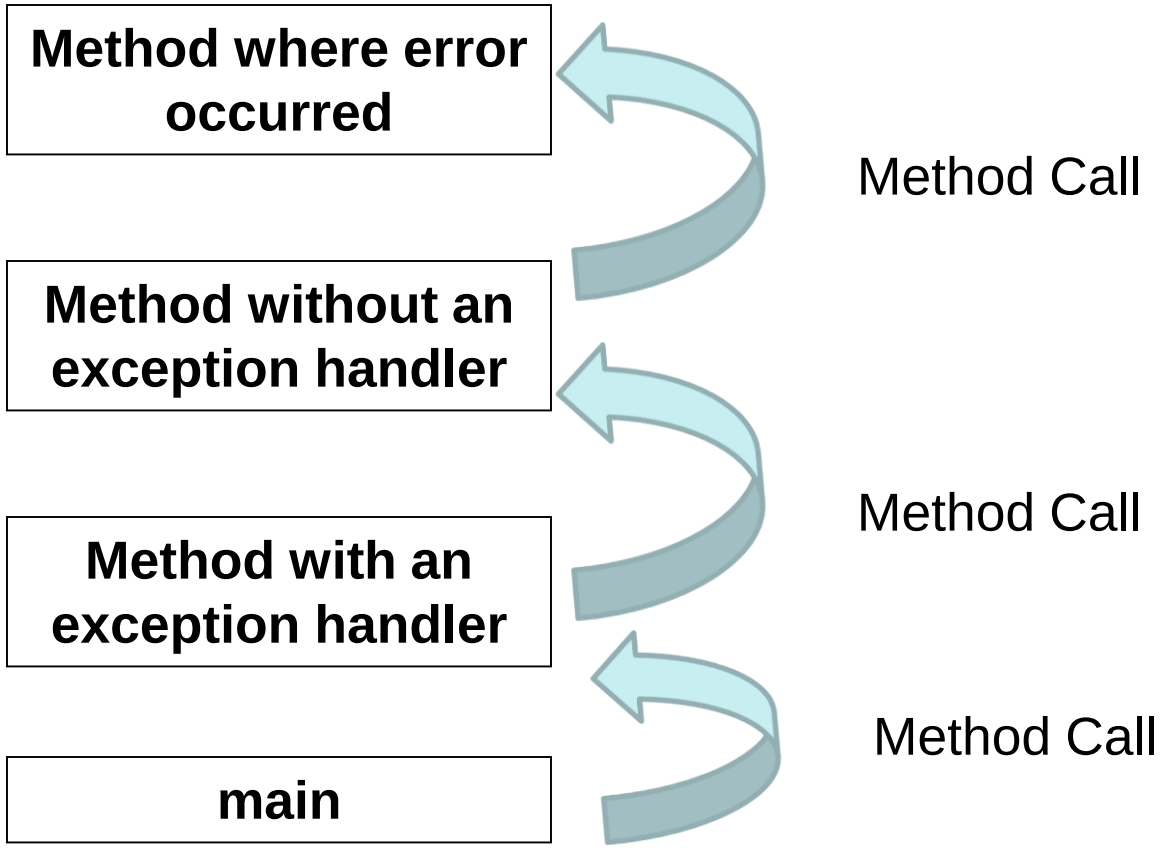
- Drawbacks of traditional error handling:
  - Most of the code deals with error handling.
  - It is hard to see the normal flow of the program.
  - The return value is “occupied” for error-handling.

- Exception Handling
  - The “normal flow” of the program and error handling code are separated.

What Is an Exception?

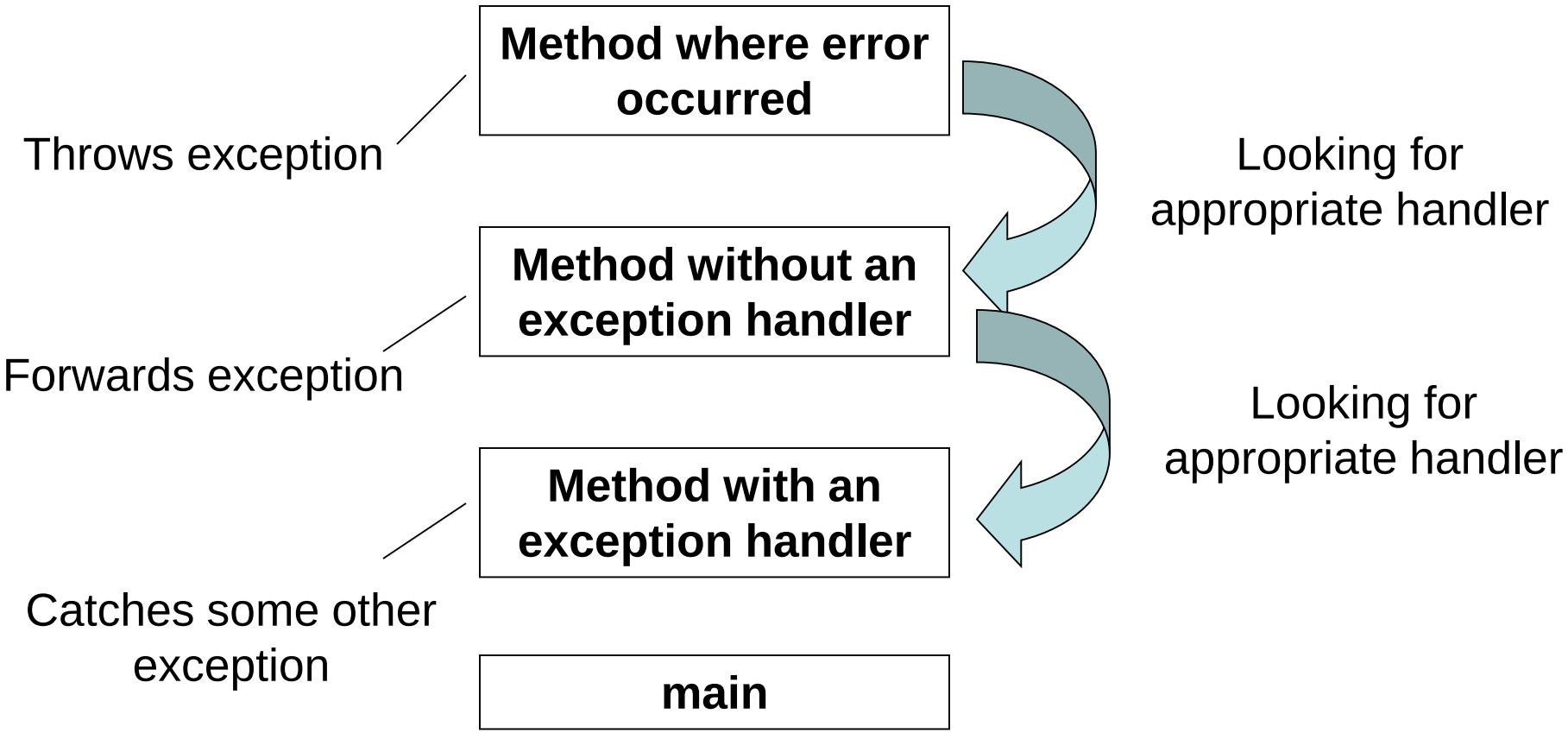
- An **exception** (“exceptional event”) is an event, which occurs during the execution of a program that disrupts the normal flow of the program’s instructions.
- When an error occurs within a method, the method creates an object and hands it off to the runtime system. The **object (“exception object”)** contains information about the error.
- Creating an exception object and handing it to the runtime system is called “**throwing an exception**”

What Is an Exception?



The Call Stack

What Is an Exception?



Searching the call stack for the exception handler

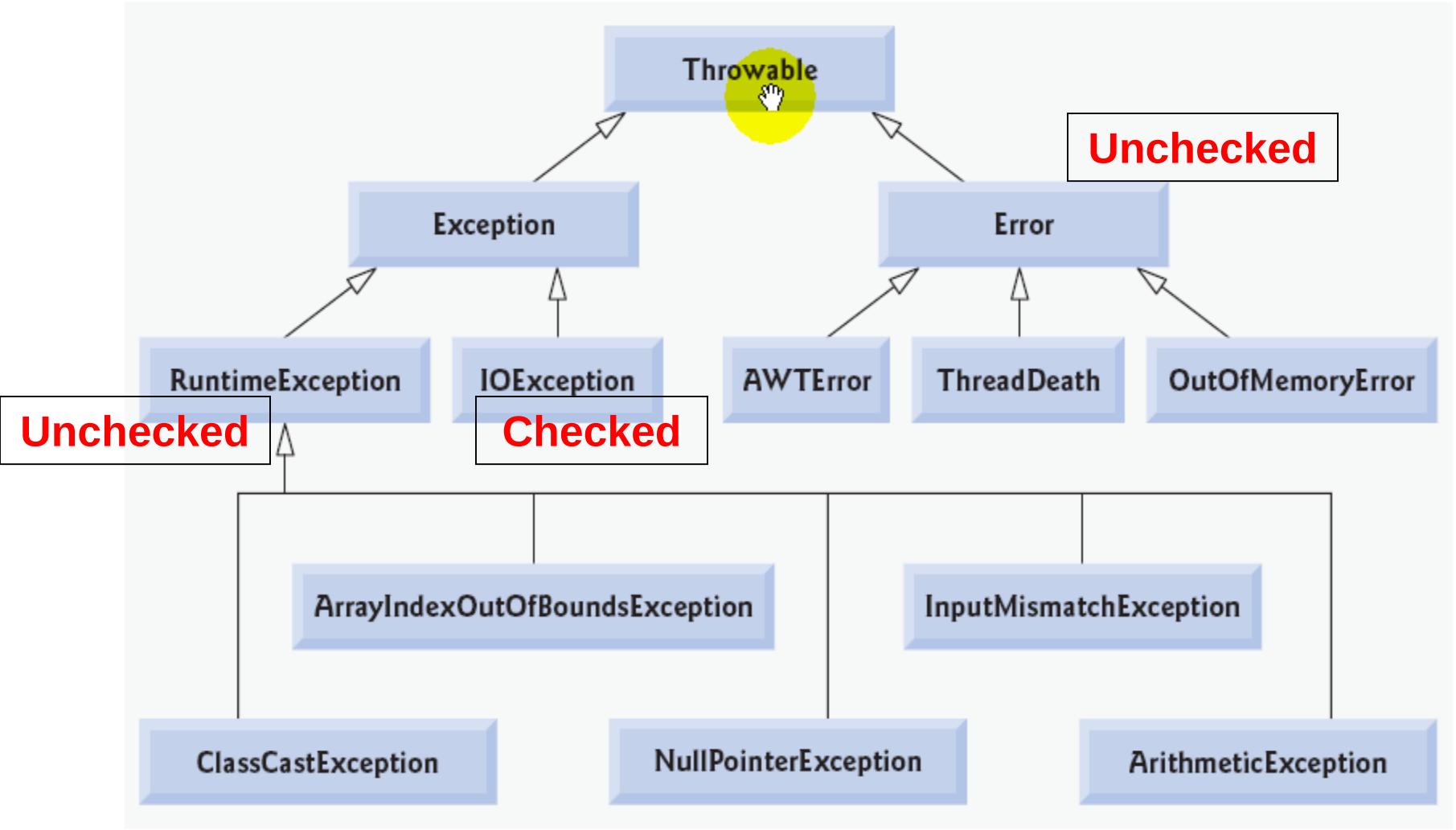
The Catch and Specify Requirement

- Catch
  - A method can catch an exception by providing an exception handler
    - A *try* statement that catches the exception The *try* must provide the handler for the exception
- Specify
  - If a method chooses not to catch, then it specifies which exceptions are thrown.
  - Exceptions are part of a method's public interface.
    - A method specifies that it can throw the exception. The method must provide a *throws* clause that lists the exception.

Kinds of Exceptions

- Three Kinds of Exceptions
  - Checked exception: can anticipate and recover the exception. Checked exceptions are subject to the Catch or Specify Requirement (CSR). All exceptions are checked exceptions, except for *Error*, *RuntimeException*, and their subclasses.
  - Error (unchecked exception): cannot anticipate and recover, not subject to CSR, ex) system malfunction
  - Runtime exception (unchecked exception): cannot anticipate and recover, not subject to CSR, ex) logic error or improper use of an API

Kinds of Exceptions



Exception Handling Keywords

- **try** – Entering a clause where exceptions are expected.
- **catch** – Caught an exception.
- **throw** – Found a problem, throwing exception.
- **throws** – List of exceptions that the method might throw.
- **finally** – Do always.

Catching and Handling Exceptions

The try, catch, and finally block

```
try {
  // try block
}
catch (ExceptionType1 param1) {
  // Exception Block
}
catch (ExceptionType2 param2) {
  // Exception Block
}
 ……
catch (ExceptionTypeN paramN) {
  // Exception Block
}
finally {
  // finally Block
}
```

Statements that have some possibilities to generate exception(s).

Execute statements here when the corresponding exception occurred.

Do always

If a finally clause is present with a try, its code is executed after all other processing in the try is complete. It allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break.

Example 1

```
import java.io.*;
public class ExcepTest{
   public static void main(String args[]){
     int a[] = {347, 975};
     try{
        System.out.println("Access element three :" + a[3]); // a[3] does not exist
        System.out.println("a[0]=" + a[0]);
      } catch(ArrayIndexOutOfBoundsException e){
        System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the try block");
   }
}
```

Output of this program:

Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3

Out of the try block

Comments on the Previous Slide

- This program tries to access element a[3]
  - System.out.println("Access element three :" + a[3]); // a[3] does not exist
- An exception is thrown.
- The program continues to run from
  - }catch(ArrayIndexOutOfBoundsException e){
- There is no execution for
  - System.out.println("Access element three :" + a[3]); // a[3] does not exist
  - System.out.println("a[0]=" + a[0]);
- 3 in the output of the program means the value of the index for a[3].

Example

```
1.  import java.io.*;
2.  public class TestException {
3.   public void noNeedException() {          // No need the exception handling
4.     int i = 100;
5.     System.out.println("i = " + 100);   }
6.   public void useTryCatch() {
7.     String name ="";
8.     System.out.print("What is your name? ");    // The code needs exception handling
9.     try {
10.       BufferedReader charStream = new BufferedReader (new InputStreamReader(System.in));
11.       name = charStream.readLine().trim();
12.       System.out.println("Your name is " + name);
13.    } catch(Exception e) {
14.        System.out.println(“Exception: " + e );     }
15.    System.out.println(“End of  the TryCatch method");    }
16.  public static void main(String[] args)  {
17.    TestException obj = new TestException();
18.    obj.noNeedException();
19.    obj.useTryCatch();
20. }
21. }
```

Comments on the example

- The program try to read an input stream of characters (lines 10 and 11).
- If the try block completed successfully (no exceptions) then: continue to line 15.
  - Lines 13 and 14 are skipped.
- If an exception was thrown then: execute the catch block (lines 13 and 14), and then continue to line 15. The try block is aborted.
  - To simulate this exception, you may type Ctrl/z or Ctrl/d keys (depending on your OS) on your keyboard when running the program.

Keyword: finally

- **finally** – This clause is executed always, either if an exception is thrown or not.
- The **finally** block always comes after the last **catch** block.

Example

```
import java.io.*;
import java.util.*;
public class ExcepTestFinally{
   public static void main(String args[]){
      int a[] = {347, 975};
      try{
         Scanner keyboard = new Scanner(System.in);        // data on the keyboard
         System.out.print("Please type an index value: ");
         int i = keyboard.nextInt();          // getting an integer value from the keyboard
         System.out.println("Access element  "+ i + ":" + a[i]);
         System.out.println("a[0]=" + a[0]);
      } catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Exception thrown  :" + e);
      } finally{
         a[0] = 6;
         System.out.println("The finally statement is executed.  a[0]= " +a[0 ]);
      }
      System.out.println("Out of the try block");
   }
}          // Output of this program see on the next slide
```

Comments on the Previous Slide

- Output of the program:
  - text in red typed by the user; text in blue printed by the program

```
java ExceptionTestFinally
Please type an index value: 3
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
The finally clause is executed. a[0]=6
Out of the try block
```

- Another run of the program:

```
java ExceptionTestFinally
Please type an index value: 1
Access element 1 :975
a[0]=347
The finally clause is executed. a[0]=6
Out of the try block
```

Comments on the Previous Slide

- The first run of the program (with the inputted value = 3) resulted in throwing the exception and execution of the finally clause.
- The second run (with the inputted value=1) resulted in the normal execution of the program and execution of the *finally* clause.

Keyword: finally

```
1.    void mergeFiles( String f1, String f2, String f3)  {
2.          try  {
3.            data1 = readFile (f1) ;                  //  step 1
4.            data2 = readFile (f2) ;                  //  step2
5.            data3 = merge (data1,  data2) ;          //   step3
6.            writeFile (f3,    data3);                // step4
7.        } catch (FileException e)  {
8.            / /  handle the exception
9.        }  catch (FormatException e)  {
10.           / /  handle the exception
11.       }   finally  {
12.           cleanup ( ) ;
13.       }
14.   }
```

try, catch, and finally

The finally clause is used to clean up internal state or to release non-object resources, such as open files stored in local variables.

```
public boolean searchFor (String file, String word) throws StreamException
{
  Stream input = null;
  try {
      input = new Stream(file);
      while (!input.eof())
        if (input.next().equals(word)) return true;
      return false;       // not found
   } finally {
         if (input != null) input.close();
   }
}
```

Summary

- The Java language uses exceptions to provide for handling errors.
- One advantage of using exceptions is that compiler will check to see if a possible error is being checked for.
- Regular code and error handling code are separated.
- A thrown exception can be caught by the caller of the method.
- Exception raising and handling keywords are: *try*, *catch*, *throw*, *throws*, and *finally*.