Chapter Seven

Exceptions Handling

Exceptions

- An exception is an object that describes an unusual or erroneous situation
- Exceptions are thrown by a program, and may be caught and handled by another part of the program

Exception Hierarchy

- **Errors**:
  - Errors are fairly rare and usually fatal. They are caused by bugs in the Java VM, or by the program running out of memory or other resources.
  - Errors are usually not handled by programs.
- **Exceptions**:
  - Exceptions can be caused by bugs in the program or improper data supplied to the program
  - Exceptions should be handled, or caught
  - An exception is either checked or unchecked
  - Checked exceptions must be caught or declared as thrown.
  - You are not required to handle unchecked exceptions.

Attack of the Exception

```
public static int average(int[] a) {
     int total = 0;
     for(int i = 0; i < a.length; i++) {
          total += a[i];
     }
     return total / a.length;
}
```

- What happens when this method is used to take the average of an array of length zero?
- Program throws an *Exception* and fails

```
java.lang.ArithmeticException: / by zero
```

What is an Exception?

- An error event that disrupts the program flow and may cause a program to fail.
- Some examples:
  - Performing illegal arithmetic
  - Illegal arguments to methods
  - Accessing an out-of-bounds array element
  - Hardware failures
  - Writing to a read-only file

Another Exception Example

- What is the output of this program?

```
public class ExceptionExample {
    public static void main(String args[]) {
        String[] greek = {"Alpha", "Beta"};
        System.out.println(greek[2]);
    }
}
```

Output:

Exception in thread "main“ java.lang.ArrayIndexOutOfBoundsException: 2
    at ExceptionExample.main(ExceptionExample.java:4)

Exception Message Details

Exception message format:

[exception class]: [additional description of exception]
at [class].[method]([file]:[line number])

Example:

```
java.lang.ArrayIndexOutOfBoundsException: 2
    at ExceptionExample.main(ExceptionExample.java:4)
```

- What exception class? `ArrayIndexOutOfBoundsException`
- Which array index is out of bounds? `2`
- What method throws the exception? `main`
- What file contains the method? `ExceptionExample.java`
- What line of the file throws the exception? `4`

Classifying Java Exceptions

- **Unchecked Exceptions**
  - All the exceptions we've seen so far have been *Unchecked Exceptions*, or *Runtime Exceptions*
  - Usually occur because of programming errors, when code is not robust enough to prevent them
  - They are numerous and can be ignored by the programmer
  - **Common Unchecked Exceptions**
    - **NullPointerException**: reference is null and should not be
    - **IllegalArgumentException**: method argument is improper is some way
    - **IllegalStateException**: method called when class is in improper state

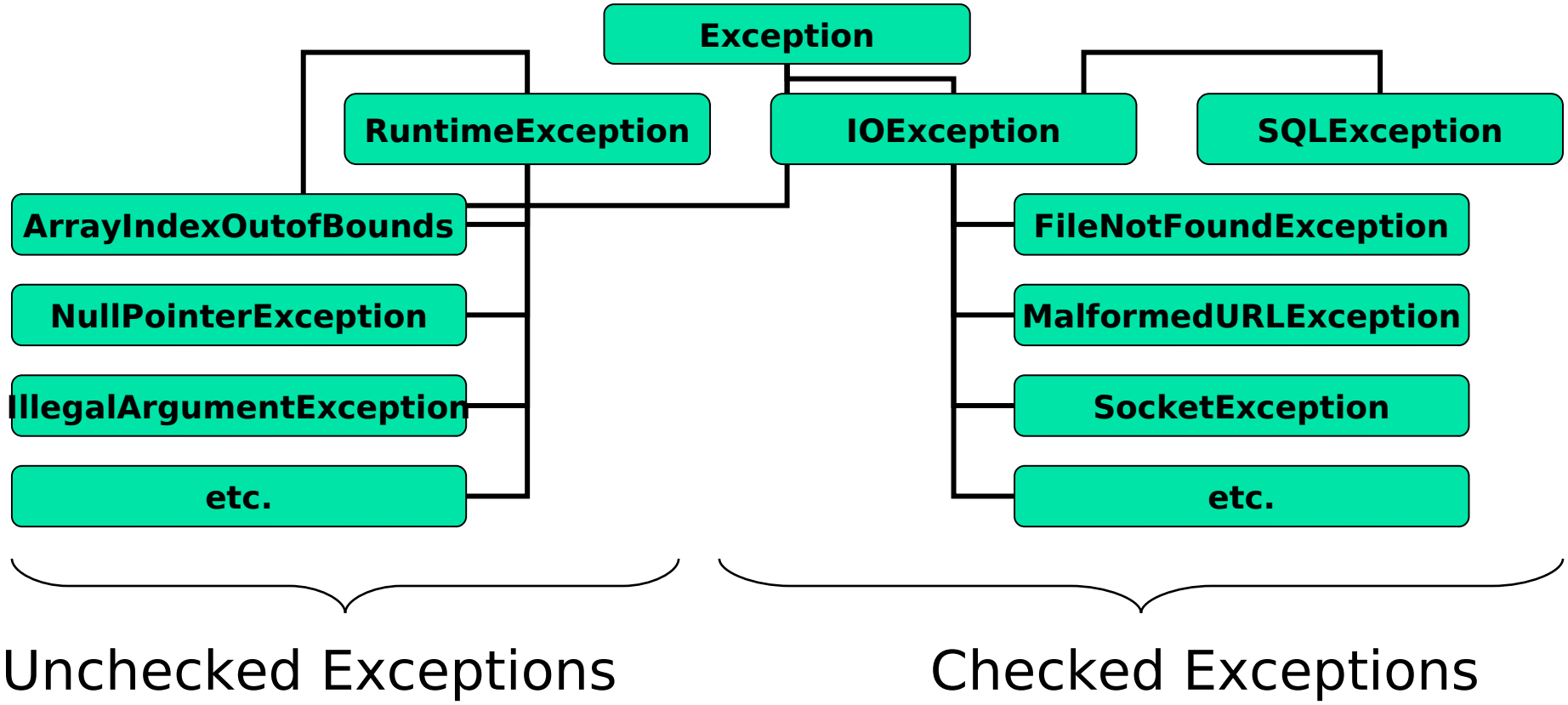
- **Checked Exceptions**
  - Usually occur because of errors programmer cannot control
    - **examples**: hardware failures, unreadable files
  - They are less frequent and they cannot be ignored by the programmer . . .
  - Every method must catch (handle) checked exceptions or specify that it may throw them
  - Specify with the **`throws`** keyword

Checked and Unchecked Exceptions

Checked Exception	**Unchecked Exception**
not subclass of RuntimeException	subclass of RuntimeException
if not caught, method *must* specify it to be thrown	if not caught, method *may* specify it to be thrown
for errors that the programmer *cannot* directly prevent from occurring	for errors that the programmer *can* directly prevent from occurring,
IOException, **FileNotFoundException**, **SocketException**	**NullPointerException**, **IllegalArgumentException**, **IllegalStateException**

Exception Class Hierarchy

- All exceptions are instances of classes that are subclasses of Exception



Keywords for Java Exceptions

- **throws: -** Describes the exceptions which can be raised by a method.
- **throw: -** Raises an exception to the first available handler in the call stack, unwinding the stack along the way.
- **try: -** Marks the start of a block associated with a set of exception handlers.
- **catch: -** If the block enclosed by the try generates an exception of this type, control moves here; watch out for implicit subsumption.
- **finally: -** *Always* called when the try block concludes, and after any necessary catch handler is complete.

Exceptions Terminology

- When an exception happens we say it was **thrown** or **raised**
- When an exception is dealt with, we say the exception is **handled** or **caught**

Exception Handling

- Use a **try-catch** block to handle exceptions that are thrown

```
try {
    // code that might throw exception
}
catch ([Type of Exception] e) {
    // what to do if exception is thrown
}
```

Exception Handling Example

```
public static int average(int[] a) {
        int total = 0;
        for(int i = 0; i < a.length; i++) {
                total += a[i];
        }
        return total / a.length;
}

public static void printAverage(int[] a) {
        try {
                int avg = average(a);
                System.out.println("the average is: " + avg);
        }
        catch (ArithmeticException e) {
                System.out.println("error calculating average");
        }
}
```

Catching Multiple Exceptions

- Handle multiple possible exceptions by multiple successive catch blocks

```
try {
    // code that might throw multiple exception
}
catch (IOException e) {
    // handle IOException and all subclasses
}
catch (ClassNotFoundException e2) {
    // handle ClassNotFoundException
}
```

Exception Constructors

- Exceptions have at least two constructors:

1. no arguments

   NullPointerException e = new NullPointerException();

2. single String argument *descriptive message that appears when exception error message is printed*

   IllegalArgumentExceptione e =

new IllegalArgumentException("number must be positive");

Writing Your Own Exception

- To write your own exception, write a subclass of Exception and write both types of constructors

```
public class MyCheckedException extends IOException {
    public MyCheckedException() {}
    public MyCheckedException(String m) {
            super(m);
    }
}
public class MyUncheckedException extends RuntimeException {
    public MyUncheckedException() {}
    public MyUncheckedException(String m) {
            super(m);
   }
}
```

Throwing Exceptions

- Throw exception with the **throw** keyword

```
public static int average(int[] a) {
        if (a.length == 0) {
                throw new IllegalArgumentException("array is empty");
        }
        int total = 0;
        for(int i = 0; i < a.length; i++) {
                total += a[i];
        }
        return total / a.length;
}
```

Throw Statement and Throws Clause

```
public class MyClass {
        public void aMethod(int i, int j) throws MyException {
                // ...
                 throw new MyException(reason);
                // ...
        }
}
```

Throws and Inheritance

- A method can throw less exceptions, but not more, than the method it is overriding

```
public class MyClass {
    public void doSomething() throws IOException, SQLException {
            // do something here
    }
}

public class MySubclass extends MyClass {
    public void doSomething() throws IOException {
            // do something here
    }
}
```