FUNDAMENTALS

CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

6: Exceptions & I/O

Alexandra Papoutsaki
LECTURES

Mark Kampe
LABS

Lecture 6: Exceptions & I/O

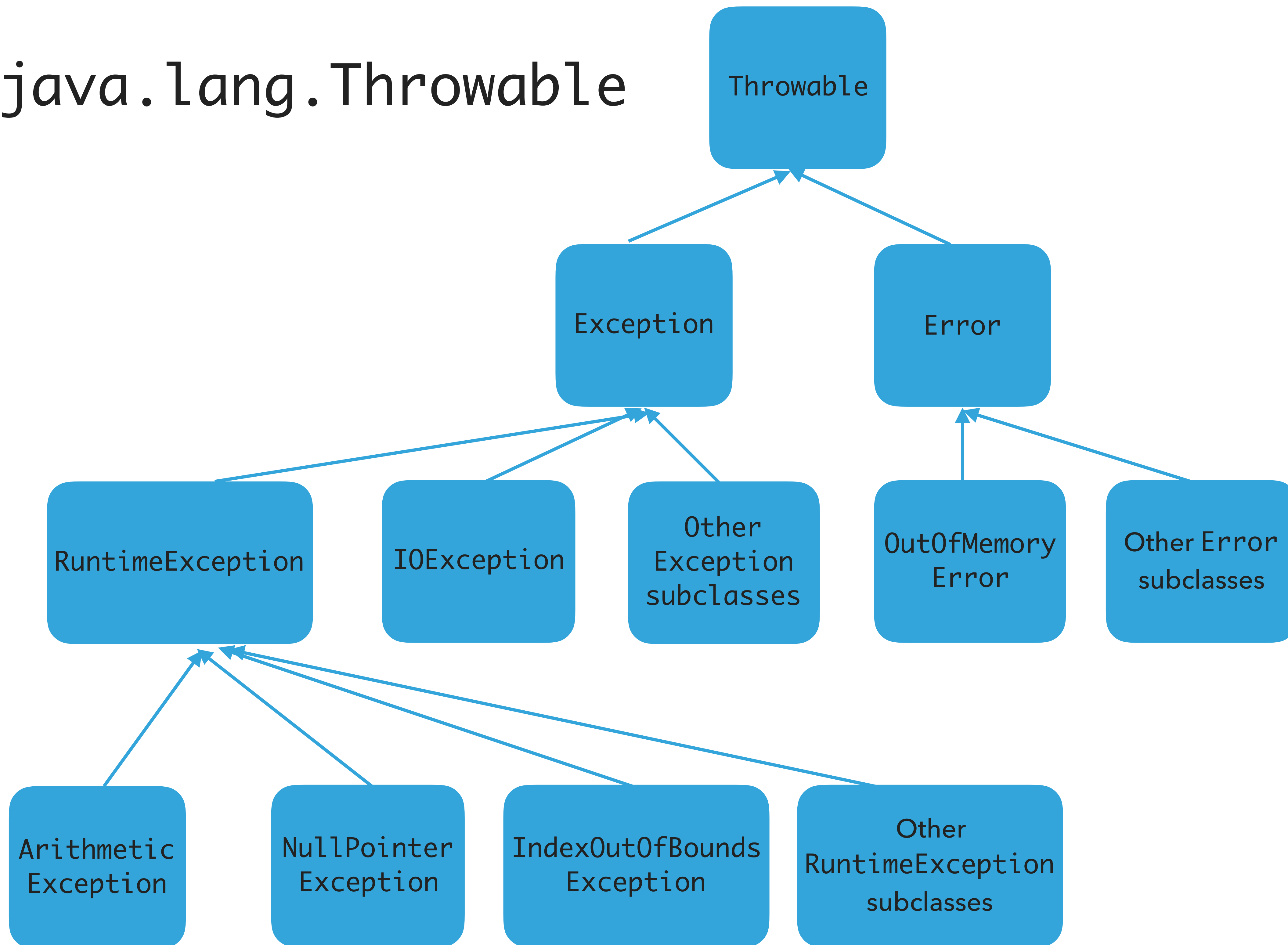
- Exceptions
- Assertions
- Text I/O
- Binary I/O

Some slides adopted from Introduction to Java and Oracle tutorials

Exceptions are exceptional or unwanted events

- That is operations that disrupt the normal flow of the program.
  - E.g., divide a number by zero, run out of memory, ask for a file that does not exist, etc.
- When an error occurs within a method, the method throws an exception object that contains its name, type, and state of program.
- The runtime system looks for something to handle the exception among the call stack, the list of methods called (in reverse order) by `main` to reach the error.
- The exception handler catches the exception. If no appropriate handler, the program terminates.

https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html



https://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html

Three major types of exception classes

- `Error`: rare internal system errors that an application cannot recover from.
    - Typically not caught and program has to terminate.
    - e.g., `java.lang.OutOfMemoryError` or `java.lang.StackOverflowError`
- `Exception`: errors caused by program and external circumstances.
    - Can be caught and handled.
    - e.g., `java.io.Exception`
- `RuntimeException`: programming errors that can occur in any Java method.
    - Method not required to declare that it throws any of the exception.
    - e.g., `java.lang.IndexOutOfBoundsException`, `java.lang.NullPointerException`, `java.lang.ArithmeticException`
- Unchecked exceptions: `Error` and `RuntimeException` and subclasses.
- `Checked exceptions`: All other exceptions - programmer has to check and deal with them.

https://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html

Handling exceptions

- Three operations:
  - Declaring an exception
  - Throwing an exception
  - Catching an exception

```
method1(){
    try {
        method2();
    } catch (Exception e) {
            System.err.println(e.getMessage());
    }
}
method2() throws Exception{
    if(some error) {
        throw new Exception();
    }
}
```

CATCH EXCEPTION

DECLARE EXCEPTION

THROW EXCEPTION

https://docs.oracle.com/javase/tutorial/essential/exceptions/catch.html

Declaring exceptions

- Every method must state the types of *checked* exceptions it might throw in the method header so that the caller of the method is informed of the exception.
  - System errors and runtime exceptions can happen to any code, therefore Java does not require explicit declaration of unchecked exceptions.
- `public void exceptionalMethod() throws IOException{`
- `throws`: the method might throw an exception. Can also throw multiple exceptions, separated by comma.

https://docs.oracle.com/javase/specs/jls/se7/html/jls-8.html#jls-8.4.6

Throwing exceptions

- If an error is detected, then the program can throw an exception.
  - e.g., you have asked for age and the user gave you a string. You can throw an `IllegalArgumentException`.
- `throw new IllegalArgumentException("Wrong argument");`
  - The argument in the constructor is called the exception message. You can access it by invoking `getMessage()`.
- `throws` **FOR DECLARING AN EXCEPTION,** `throw` **TO THROW AN EXCEPTION**

https://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html

Catching exceptions

- An exception can be caught and handled in a try-catch block.

```
method(){
    try {
         statements; //statements that could thrown exception
    } catch (Exception1 e1) {
            //handle e1;
    }
    catch (Exception2 e2) {
            //handle e2;
    }
}
```

- If no exception is thrown, then the catch blocks are skipped.
- If an exception is thrown, the execution of the try block ends at the responsible statement.
- The order of catch blocks is important. A compile error will result if a catch block for a superclass type appears before a catch block for a subclass. E.g., catch(Exception ex) followed by catch(RuntimeException ex) won't compile.
- If a method declares a checked exception (e.g., `void p1() throws IOException`) and you invoke it, you have to enclose it in a try catch block or declare to throw the exception in the calling method (e.g., `try{ p1();} catch (IOException e){…}`.

https://docs.oracle.com/javase/specs/jls/se7/html/jls-8.html#jls-8.4.6

```
main method{
   …
   try{
          …
          method1();
          statement1;
    }
    catch (Exception1 ex1){
       //process ex1
    }
    statement2;
}
```

```
method1{
   …
   try{
          …
          method2();
          statement3;
    }
    catch (Exception2 ex2){
       //process ex2
    }
    statement4;
}
```

```
method2{
   …
   try{
          …
          method3();
          statement5;
    }
    catch (Exception3 ex3){
       //process ex3
    }
    statement6;
}
```

Assume `method3` throws an exception. Possible outcomes:

- Exception is of type `Exception3`. Caught in `method2`. `statement5` is skipped. `statement6` is executed.
- Exception is of type `Exception2`. Caught in `method1`. `statement3` is skipped. `statement4` is executed.
- Exception is of type `Exception1`. Caught in `main`. `statement1` is skipped. `statement2` is executed.
- Exception is not caught in `method2`, `method1`, and `main`, the program terminates. `statement1` and `statement2` are not executed.

Example from Introduction to Java Programming by Liang

finally block

- Used when you want to execute some code regardless of whether an exception occurs or is caught

```
method(){
    try {
        statements; //statements that could thrown exception
    } catch (Exception1 e) {
            //handle e; catch is optional.
    }
    finally{
            //statements that are executed no matter what;
    }
}
```

- The `finally` block will execute no matter what. Even after a `return`.

https://docs.oracle.com/javase/tutorial/essential/exceptions/declaring.html

```java
/**
 * Illustrates try,catch, finally blocks
 * @author https://docs.oracle.com/javase/tutorial/essential/exceptions/putItTogether.html
 *
 */
import java.io.*;
import java.util.List;
import java.util.ArrayList;

public class ListOfNumbers {
	// Note: This class will not compile yet.

	private List<Integer> list;
	private static final int SIZE = 10;

	public ListOfNumbers() {
		list = new ArrayList<Integer>(SIZE);
		for (int i = 0; i < SIZE; i++) {
			list.add(new Integer(i));
		}
	}

	public void writeList() {
		PrintWriter out = null;

		try {
			System.out.println("Entering" + " try statement");

			out = new PrintWriter(new FileWriter("OutFile.txt"));
			for (int i = 0; i < SIZE; i++) {
				out.println("Value at: " + i + " = " + list.get(i));
			}
		} catch (IndexOutOfBoundsException e) {
			System.err.println("Caught IndexOutOfBoundsException: " + e.getMessage());

		} catch (IOException e) {
			System.err.println("Caught IOException: " + e.getMessage());

		} finally {
			if (out != null) {
				System.out.println("Closing PrintWriter");
				out.close();
			} else {
				System.out.println("PrintWriter not open");
			}
		}
	}

}
```

Practice Time

- 1. Is there anything wrong with this exception handler?

```
try {

} catch (Exception e) {

} catch (ArithmeticException a) {

}
```

Answers

- 1. The ordering matters! The second handler can never be reached and the code won't compile.

https://docs.oracle.com/javase/tutorial/essential/exceptions/QandE/answers.html

Lecture 6: Exceptions & I/O

- Exceptions
- Assertions
- Text I/O
- Binary I/O

Pre and post conditions

- Pre-condition: Specification of what must be true for method to work properly.
- Post-condition: Specification of what must be true at end of method if precondition held before execution.

https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

Assertions test correctness of assumptions about our program

- An assertion must be a statement that is either true or false and should be true if there are no mistakes in the program.
- Two forms:

```
assert booleanExpression ;
assert booleanExpression : message ;
```

- If they evaluate to true, nothing happens.
- If they fail, they throw an AssertionError.
- E.g., `assert  age >= 21 : " Underage”;`
- If failed:
  - `Exception in thread "main" java.lang.AssertionError: Underage`

https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

Enabling assertions

- By default off.
  - `java -ea`
  - Or adding `ea` as virtual machine argument in arguments tab in Eclipse when set up runtime configuration.
- Little cost as they can be turned on/off.
- That means that they should NOT be used to check arguments in public methods.
  - **USE EXCEPTIONS INSTEAD!**

https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

Lecture 6: Exceptions & I/O

- Exceptions
- Assertions
- Text I/O
- Binary I/O

I/O streams

- Input stream: a sequence of data into the program.
- Output stream: a sequence of data out of the program.
- Stream sources and destinations include disk files, keyboard, peripherals, memory arrays, other programs, etc.
- Data stored in variables, objects and data structures are temporary and lost when the program terminates. Streams allow us to save them in files, e.g., on disk or CD (!)
- Streams can support different kinds of data: bytes, principles, characters, objects, etc.

https://docs.oracle.com/javase/tutorial/essential/io/streams.html

Text and Binary files

- Text files: Contain sequences of characters and can be viewed in a text editor or read by a program.
  - Typically set to ASCII encoding.
  - Common extension: .txt
- Binary files: Contents are handled as sequences of binary digits by programs.
  - Common extension: .dat

https://docs.oracle.com/javase/tutorial/essential/io/charstreams.html

Files

- Every file is placed in a directory in the file system.
- Absolute file name: the file name with its complete path and drive letter.
  - e.g., on Windows: `C:\apapoutsaki\somefile.txt`
  - On Mac/Unix: `/home/apapoutsaki.somefile.txt`
- `File`: contains methods for obtaining file properties, renaming, and deleting files. Not for reading/writing!
- **CAUTION: DIRECTORY SEPARATOR IN WINDOWS IS \, WHICH IS SPECIAL CHARACTER IN JAVA. SHOULD BE "\\" INSTEAD.**

TEXT I/O

```java
/**
 * Demonstrates File class and its operations.
 * @author https://liveexample.pearsoncmg.com/html/TestFileClass.html
 *
 */

import java.io.File;
import java.util.Date;

public class TestFile {
  public static void main(String[] args) {
    File file = new File("some.text");
    System.out.println("Does it exist? " + file.exists());
    System.out.println("The file has " + file.length() + " bytes");
    System.out.println("Can it be read? " + file.canRead());
    System.out.println("Can it be written? " + file.canWrite());
    System.out.println("Is it a directory? " + file.isDirectory());
    System.out.println("Is it a file? " + file.isFile());
    System.out.println("Is it absolute? " + file.isAbsolute());
    System.out.println("Is it hidden? " + file.isHidden());
    System.out.println("Absolute path is " + file.getAbsolutePath());
    System.out.println("Last modified on " + new Date(file.lastModified()));
  }
}
```

Writing data to a text file

- `PrintWriter output = new PrintWriter(new File("filename"));`
- New file will be created. If already exists, discard.
- Invoking the constructor may throw an I/O Exception…
- `output.print` and `output.println` work with Strings, and primitives.
- Always close a stream!

TEXT I/O

```java
/**
 * Demonstrates how to write to text file.
 * @author https://liveexample.pearsoncmg.com/html/WriteData.html
 *
 */

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;

public class WriteData {
    public static void main(String[] args) {

        PrintWriter output = null;
        try {
            output = new PrintWriter(new File("addresses.txt"));
            // Write formatted output to the file
            output.print("Alexandra Papoutsaki ");
            output.println(222);
            output.print("Mark Kampe ");
            output.println(212);

        } catch (IOException e) {
            System.err.println(e.getMessage());
        } finally {
            if (output != null)
                output.close();
        }
    }
}
```

Reading data from a text file

- `java.util.Scanner` reads Strings and primitives.
- Breaks input into tokens, demoted by whitespaces.
- To read from keyboard: `Scanner input = new Scanner(System.in);`
- To read from file: `Scanner input = new Scanner(new File("filename"));`
- Need to close stream as before.
- `hasNext()` tells us if there are more tokens in the stream. `next()` returns one token at a time.
    - Variations of next are `nextLine()`, `nextByte()`, `nextShort()`, etc.

TEXT I/O

```java
/**
 * Demonstrates how to read data from a text file.
 * @author https://liveexample.pearsoncmg.com/html/ReadData.html
 *
 */

import java.io.File;
import java.io.IOException;
import java.util.Scanner;

public class ReadData {
    public static void main(String[] args) {

        Scanner input = null;
        // Create a Scanner for the file
        try {
            input = new Scanner(new File("addresses.txt"));

            // Read data from a file
            while (input.hasNext()) {
                String firstName = input.next();
                String lastName = input.next();
                int room = input.nextInt();
                System.out.println(firstName + " " + lastName + " " + room);
            }
        } catch (IOException e) {
            System.err.println(e.getMessage());
        } finally {
            if (input != null)
                input.close();
        }

    }
}
```

Lecture 6: Exceptions & I/O

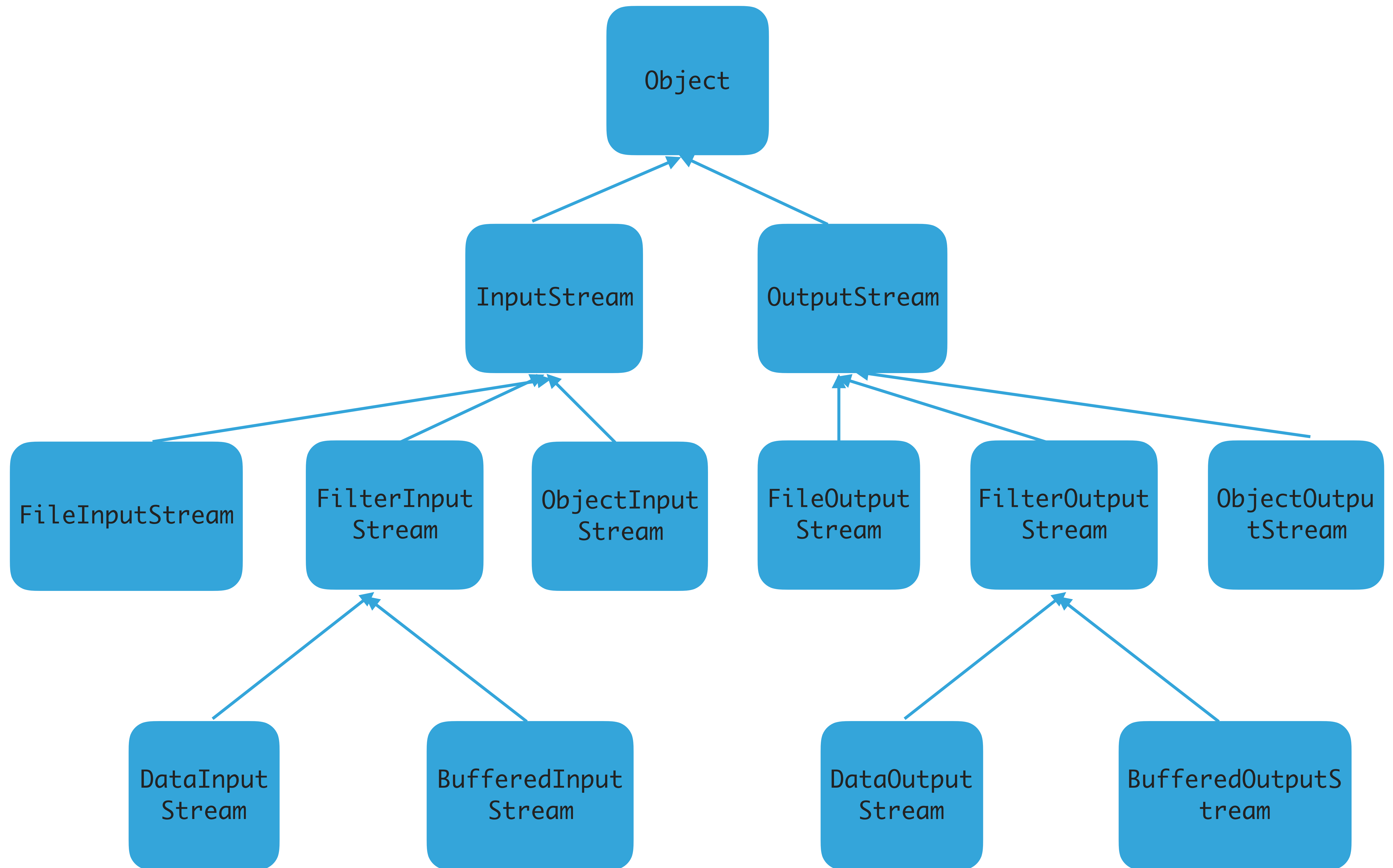
- Exceptions
- Assertions
- Text I/O
- Binary I/O

Readings:

- Oracle's guides:
  - Exceptions: https://docs.oracle.com/javase/tutorial/essential/exceptions/
  - Assertions: https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html
  - I/O: https://docs.oracle.com/javase/tutorial/essential/io
- Textbook:
  - Chapter 1.2 (Page 107)

Lecture 6: Exceptions & I/O

- Exceptions
- Assertions
- Text I/O
- Binary I/O



https://docs.oracle.com/javase/tutorial/essential/io/streams.html

Reading/Writing bytes from/to binary files.

- `FileInputStream/FileOutputStream` reads/writes bytes from/to files.
- `int read()`: reads next byte of data. Returns value between 0 to 255.
- `void write(int b)`: write next byte of data
- `close()`: closes stream

BINARY I/O

```
/**
 * Demonstrates input/output streams for binary files.
 * @author https://liveexample.pearsoncmg.com/html/TestFileStream.html
 *
 */

import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.IOException;

public class TestFileStream {
       public static void main(String[] args) throws IOException {
         try (
           // Create an output stream to the file
           FileOutputStream output = new FileOutputStream("temp.dat");
         ) {
           // Output values to the file
           for (int i = 1; i <= 10; i++)
             output.write(i);
           output.close();
         }

         try (
           // Create an input stream for the file
           FileInputStream input = new FileInputStream("temp.dat");
         ) {
           // Read values from the file
           int value;
           while ((value = input.read()) != -1)
             System.out.print(value + " ");
           input.close();
         }
       }
     }
```

Converting bytes to primitives or strings

- `DataInputStream/DataOutputStream` reads/writes bytes from/to files and converts them to appropriate type.
- Wrappers to existing input/output streams.
- `boolean/int/char/etc` `readBoolean/Int/Char/etc()`: reads a boolean/int/char/etc from an input stream.
- Void `writeBoolean/Int/Char/etc(boolean/int/char/etc)`: write a boolean/int/char/etc to an output stream.

BINARY I/O

```
/**
 * Demonstrates input/output streams for binary files.
 * @author https://liveexample.pearsoncmg.com/html/TestFileStream.html
 *
 */

import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.IOException;

public class TestFileStream {
       public static void main(String[] args) throws IOException {
         try (
           // Create an output stream to the file
           FileOutputStream output = new FileOutputStream("temp.dat");
         ) {
           // Output values to the file
           for (int i = 1; i <= 10; i++)
             output.write(i);
           output.close();
         }

         try (
           // Create an input stream for the file
           FileInputStream input = new FileInputStream("temp.dat");
         ) {
           // Read values from the file
           int value;
           while ((value = input.read()) != -1)
             System.out.print(value + " ");
           input.close();
         }
       }
     }
```

Buffered streams

- `BufferedInputStream/BufferedOutputStream` speed up read/write by using a buffer for efficient processing.
- Wrappers to existing input/output streams.
- `DataInputStream input = new DataInputStream(new FileInputStream(“temp.dat”));`
- `DataOutputStream output = new DataOutputStream(new FileOutputStream(“temp.dat”));`

Converting bytes to objects

- `ObjectInputStream/ObjectOutputStream` reads/writes bytes from/to files and converts them to
- Wrappers to existing input/output streams.
- `Object readObject()`: reads an object.
- `void readObject(Object obj)`: writes an object.