Java File I/O

CPSC 319 - Data Structures

Benyamin Bashari

Java File I/O

- Exceptions
- File I/O
  - Class Hierarchy
  - Reading
  - Writing

Exceptions

- Unexpected events in a program such as
  - Dividing by zero
  - Accessing out of bound element in an array
  - Writing to a read-only file
- Required for file handling in Java.
- Possible exceptions in the program must be handled properly, otherwise the program terminates unexpectedly. (see ExceptionTest1.java)

Exceptions (ExceptionTest1.java)

```
/**
* Read an ineteger n from System.in (number of elements)
* Read n integer numbers from System.in
* Find the Smallest number and print it
*/
public static void wrongMethod() {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int[] a = new int[n];
    for(int i = 1 ; i <= n ; i++) //This is a common mistake
        a[i] = sc.nextInt();
    int minNumber = Integer.MAX_VALUE;
    for(int i = 1 ; i <= n ; i++)
        minNumber = Math.min(minNumber, a[i]);
    System.out.println(minNumber);
}
```

- First for loop tries to access a[n], which is out of bound of the array, so program terminates with the following exception `Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException`

Types of Exceptions

- There are two types of exceptions in Java **checked** and **unchecked**
- Exceptions that are checked during compilation time are called **Checked** exceptions.
  - For example IOException (Required for find handling in Java)
  - Not handling these exceptions results to compilation error
- Exceptions that are checked during runtime are called **Unchecked** exceptions
  - For example Arithmetic Exceptions (Dividing by zero)
  - They can be ignored while coding (Not suggested, because if the exceptions happens program terminates)

Handling Exceptions (Propagation)

- Consider the following example
  - There are two methods in the program (`firstMethod()`, `secondMethod()`)
  - `main(String[] args)` calls `firstMethod()` and then `firstMethod()` calls `secondMethod()`
  - There is an exception in `secondMethod()`
  - `Runtime Stack: main()` -> `firstMethod()` -> `secondMethod()` (Exception in here)
  - If the exception is handled in the `secondMethod()` then there is no need to write any code in `main()` and `firstMethod()`
  - But if the excpetion is not handled in `secondMethod()`, then it propagates to `firstMethod()` and if the `firstMethod()` also do not handle the exception it propagates to `main()` method.
  - Eventually, if `main()` method also do not handle the exception the program terminates.

Handling Exceptions (Method 1)

- Just declare that this method might generate an exception (see ExceptionExample.java)
- Example:

```
public static int notTooSafeDivision(int a, int b) throws ArithmeticException {
        return a/b;
}
```

- This method only declare that it might throw `ArithmeticException`, so the method which calls `notTooSafeDivision` should handle the exception (or other methods in the runtime stack)

Handling Exceptions (Method 2)

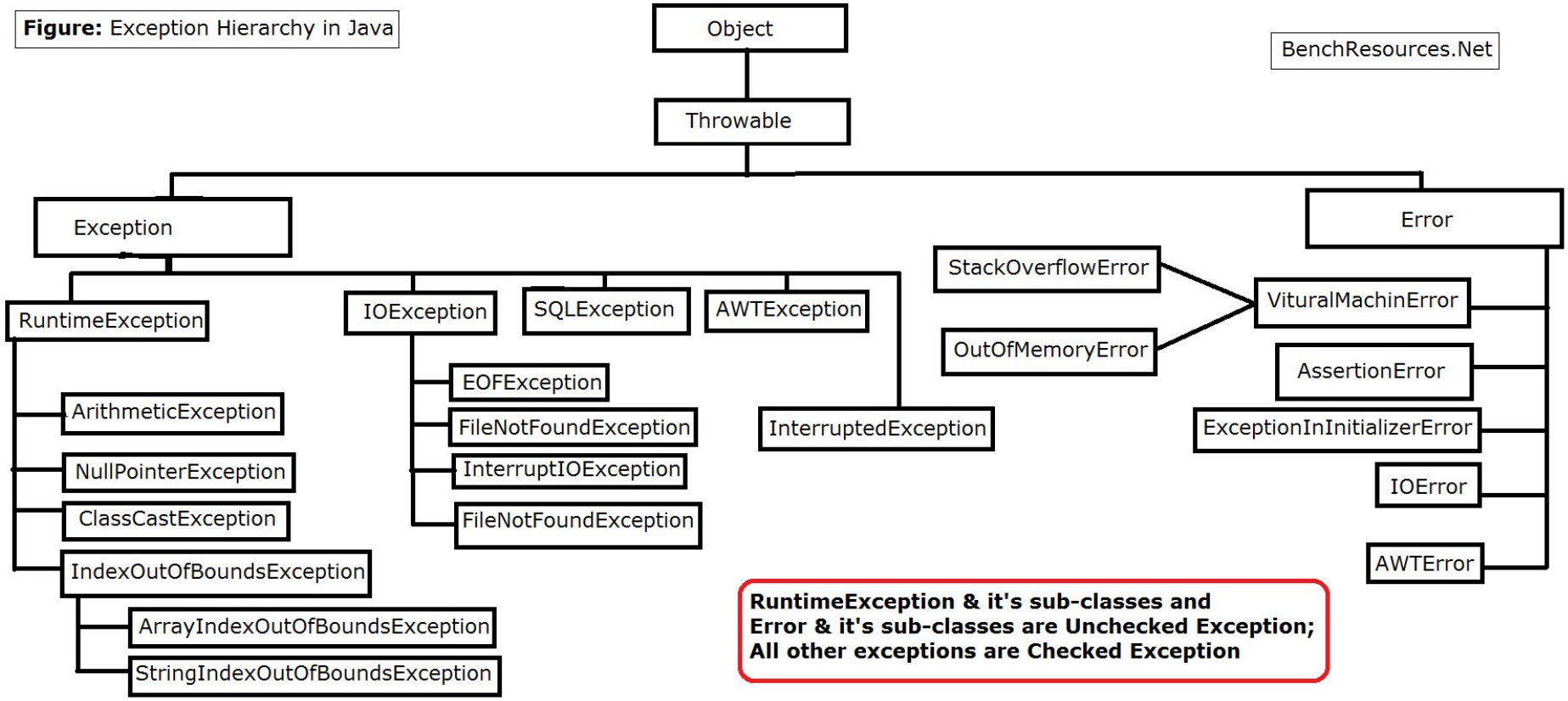
- Handling exception using try-catch-finally clause (see ExceptionExample.java)
- Example:

```
public static int safeDivision(int a, int b) {
        try {
            return a/b;
        }
        catch (ArithmeticException e) {
            System.out.println("Exception occurred");
            return 0;
        }
}
```

Handling Exceptions (Method 2)

- This example tries to divide a by b, if an `ArithmeticException` occures, then it goes to catch clause. (More than one catch clause might be used to handle the exception)
- Finally clause is used to determine what part of the code needs to be executed, whether the exceptions happen or not.

Java Exceptions Hierarchy



File Handling in Java

- File Class is used to represent a file in java
- Followings are more useful methods in File API
  - `File(String pathname)`
    - Creates a new File instance by converting the given pathname string into an abstract pathname.
  - boolean **canRead**()
    - Tests whether the application can read the file denoted by this abstract pathname.
  - boolean **canWrite**()
    - Tests whether the application can modify the file denoted by this abstract pathname.
  - boolean **createNewFile**()
    - Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.

File Handling in Java

- boolean **delete**()
  - Deletes the file or directory denoted by this abstract pathname.
- boolean **exists**()
  - Tests whether the file or directory denoted by this abstract pathname exists.
- boolean **isDirectory**()
  - Tests whether the file denoted by this abstract pathname is a directory.
- Boolean **isFile**()
  - Tests whether the file denoted by this abstract pathname is a normal file.
- Boolean **mkdir**()
  - Creates the directory named by this abstract pathname.
- Boolean **renameTo**(**File** dest)
  - Renames the file denoted by this abstract pathname.

File I/O in Java

- Reading and Writing in Java can be done on characters or binary code, the File I/O introduced here only focus on character stream in Java.
- There are different classes for Reading and Writing such as
  - Reading
    - FileReader
    - BufferedReader
    - Scanner
  - Writing
    - FileWriter
    - BufferedWriter
    - PrintWriter

File I/O in Java

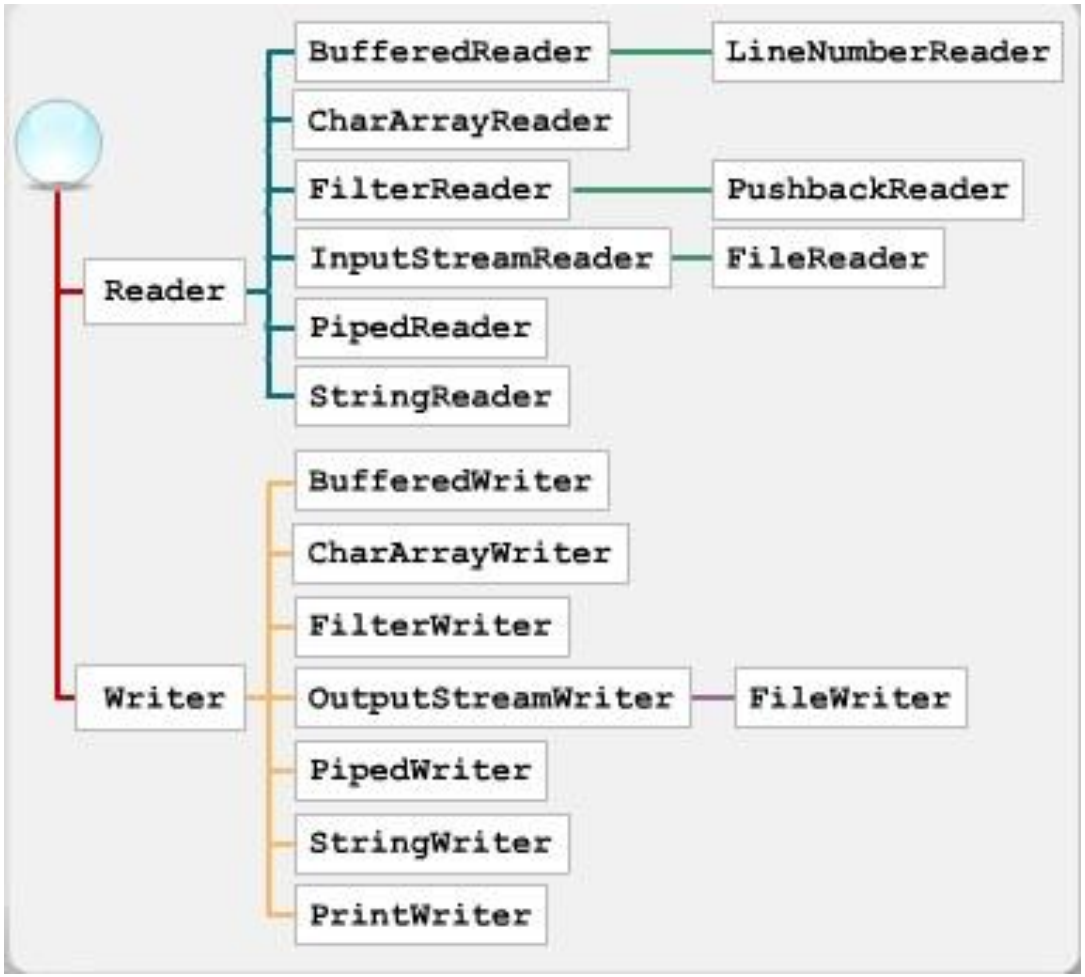


Image from http://javadevwannabe.blogspot.com/

FileReader

- Very simple class for reading streams of characters.
  - **FileReader**(**File** file)
    - Creates a new FileReader, given the File to read from.
  - public int read() throws IOException
    - Reads a single character.
    - **Returns:** The character read, or -1 if the end of the stream has been reached
  - public abstract int read(char[] cbuf, int off, int len) throws IOException
    - Reads characters into a portion of an array.
    - **Parameters:**
      - cbuf - Destination buffer
      - off - Offset at which to start storing characters
      - len - Maximum number of characters to read
    - **Returns:** The number of characters read, or -1 if the end of the stream has been reached
  - public void close() throws IOException
    - Closes the stream and releases any system resources associated with it

BufferedReader

- More efficient than FileReader
- It buffers the characters to read
- Supports the same methods of FileReader in previous slide
- Constructor
  - `BufferedReader(Reader in)`
    - Creates a buffering character-input stream that uses a default-sized input buffer.
  - FileReader can be used as the input reader (see fig in slide 14)

BufferedReader

- Other methods
  - `String readLine()`
    - Reads a line of text.
  - `Long skip(long n)`
    - Skips characters.
- See `readBufferedreader()` method, in FileExample.java

BufferedReader

```
public static void readBufferedReader() {
        try {
            File f = new File("input.txt");//input must in the same folder as FileExample.java
            if(!f.exists()) {
                System.out.println("File does not exist");
                return;
            }
            else if(!f.canRead()) {
                System.out.println("File is not readable");
                return;
            }
            BufferedReader bf = new BufferedReader(new FileReader(f));
            String s = bf.readLine();
            while(s != null) {
                System.out.println(s);
                s = bf.readLine();
            }
            bf.close();
        } catch(IOException e) {
            System.out.println("Unknow Exception");
        }
    }
```

Scanner

- Scanner can be used to read inputs from a file exactly like from System.in
  - `Scanner(File source)`
    - Constructs a new `Scanner` that produces values scanned from the specified file.
- Example

```
Scanner sc = new Scanner(f);

while(sc.hasNextLine()) {
    String s = sc.nextLine();
    System.out.println(s);
}
```

Assignment 3

- In the assignment you are asked to read inputs from a file, change it to a proper format and make BST on the words.
- String class provides useful methods to change the input to a proper format.
- First you need to remove all unwanted characters.
  - `s.replaceAll(String old, String new)` replace all substrings that are equal to old with new then returns another string.

Assignment 3

- `String line1 = line.replaceAll("[^0-9a-zA-Z ]", " ");` replace all characters which are not equal to alphabets and numbers to [space] and put the result in `line1`.
- `String line2 = line1.toLowerCase();` replace all uppercase characters to lowercase and put the result in `line2`.
- `String[] words = line2.split("\\s+");`
  - `split(String regx)` Splits the string around matches of the given regular expression.
  - When using `\s` means a whitespace character: [\t\n\x0B\f\r], using + in front of it means it can disregard multiple whitespace chars.

FileWriter

- Very simple class for writing streams of characters.
  - **FileWriter**(**File** file, boolean append)
    - Constructs a FileWriter object given a File object.
  - public void write(int c) throws IOException
    - Writes a single character.
  - public void write(String str, int off, int len) throws IOException
    - Writes a portion of a string.
  - public void flush() throws IOException
    - Flushes the stream.
  - public void close() throws IOException
    - Closes the stream, flushing it first.

BufferedWriter

- Same as FileReader and BufferedReader, there are FileWriter and BufferedWriter
- Supports the same methods of FileWriter in previous slide
- Writing is more sensitive, if proper flushing or closing of file does not happen then the content might not be written into the file.

BufferedReader

- Constructor and other method
  - `BufferedWriter(Writer out)`
    - Creates a buffered character-output stream that uses a default-sized output buffer.
    - FileWriter can be used as the input Writer (see fig in slide 14)
  - public void newLine() throws IOException
    - Writes a line separator. The line separator string is defined by the system property line.separator, and is not necessarily a single newline ('\n') character.

PrintWriter

- A convenience class for writing characters in java
- PrintWriter can be initialized with a File or a Writer
- It provides a lot of useful methods, such as printf, which is the same formatting as C printf
  - **printf**(**String** format, **Object**... args)
    - A convenience method to write a formatted string to this writer using the specified format string and arguments.
    - Same printf as C