MET CS 622: Exception Handling - Files and I/O

Reza Rawassizadeh

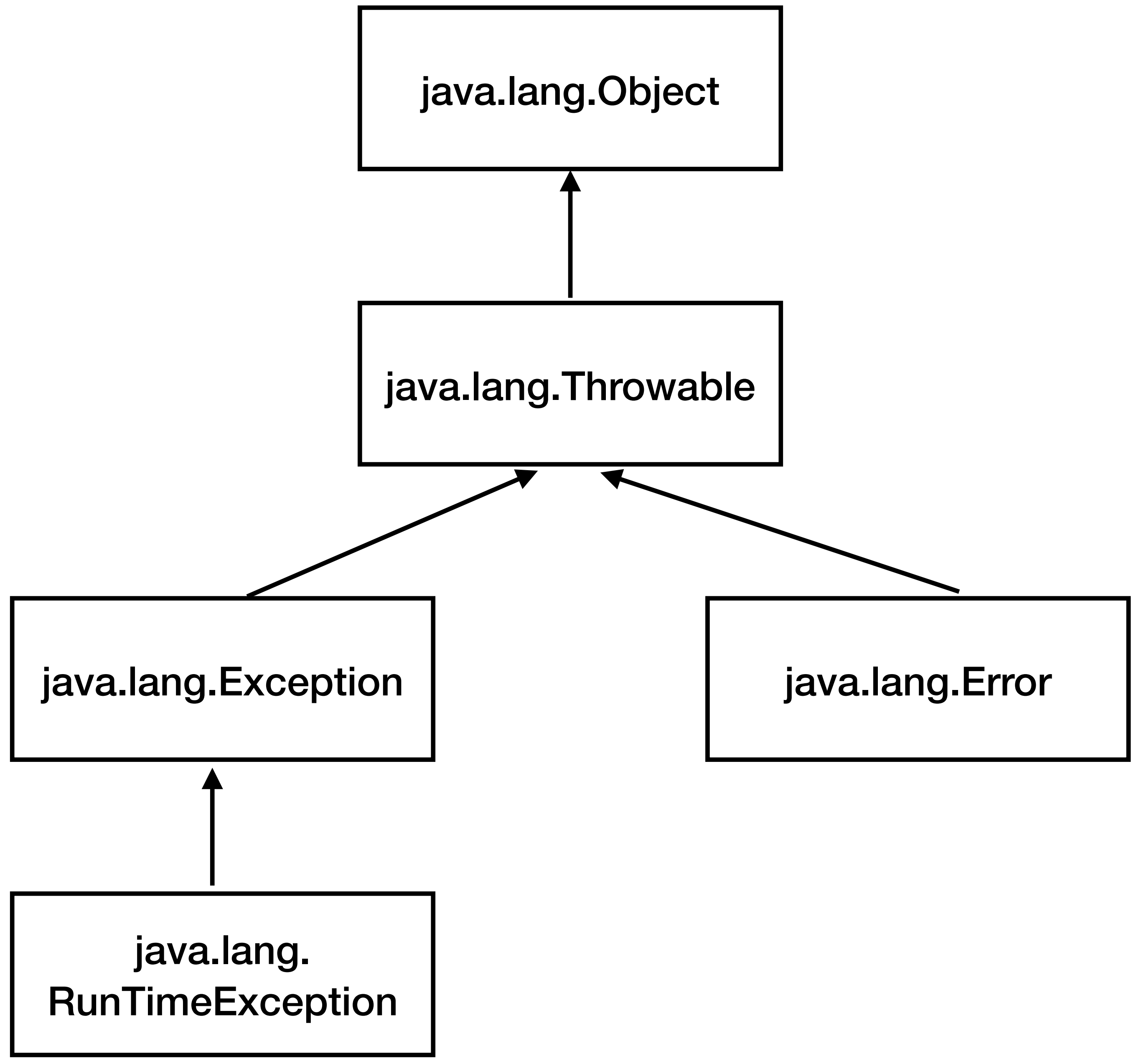
Exception Handling

- Lots of errors could occur at run time and good software should not halt its process while encountering an error.
- Exception handling is a feature used in Java to treat run time errors.
- An **exception** is a Java object that represents a condition that prevents execution from proceeding normally. If those problems are not dealt with by exception handling features, the program would crash (abort) without getting a chance to come to a natural halt.

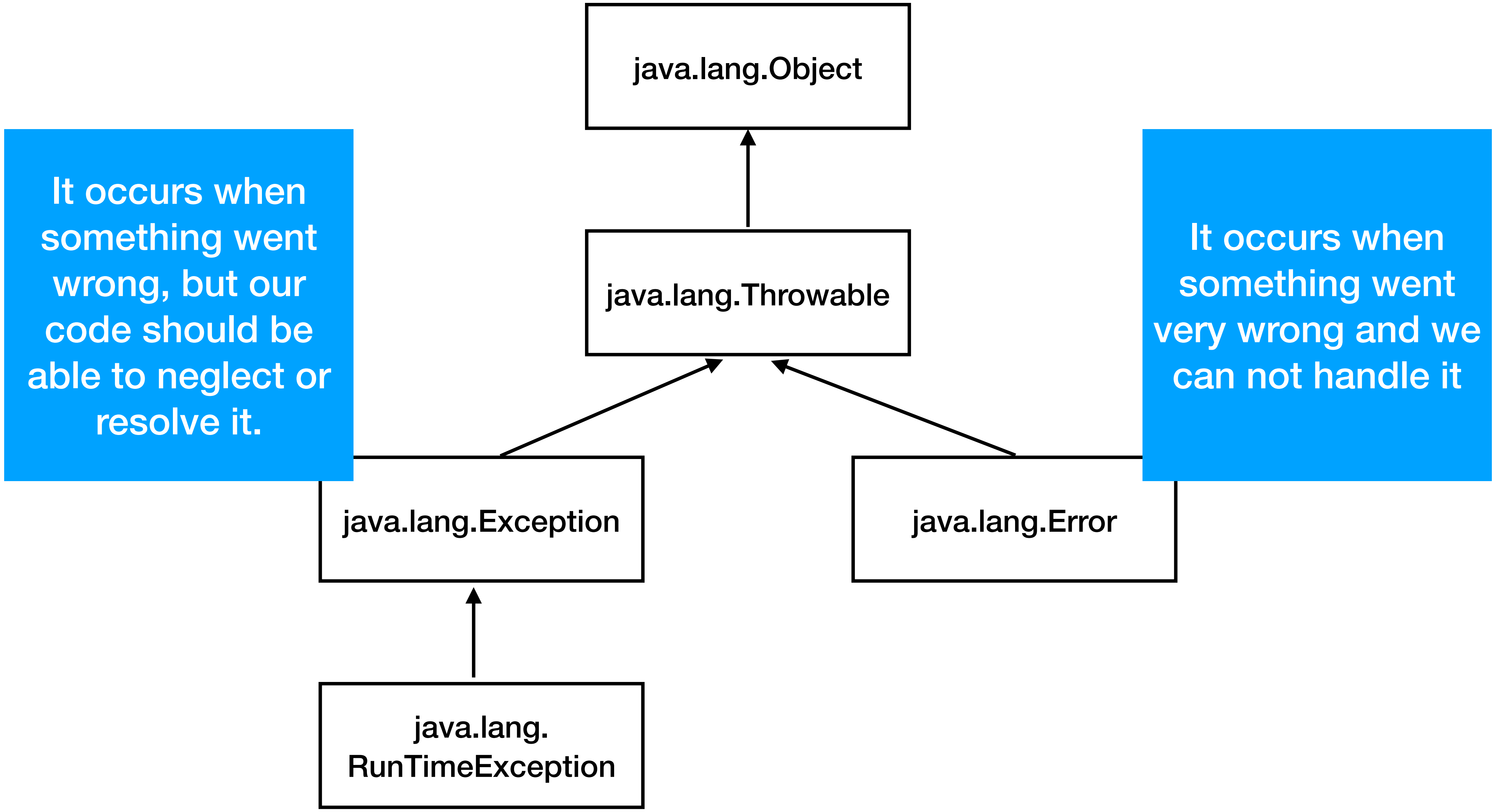
When Exception Occurs

- The code is regularly making a connection to an external web site, but that target web site is down at some point.
- Once new data entered into the system and you didn’t realize it while you are developing your codes. Therefore, an exception could raise by the java compiler.
- The IT department changes the OS access policy without informing the development team. A part of your code which accesses a file now can’t access that file anymore.
- …..

Exceptions


java.lang.Object
java.lang.Throwable
java.lang.Exception
java.lang.Error
java.lang.
RunTimeException

Exceptions



java.lang.Object
java.lang.Throwable
java.lang.Exception
java.lang.Error
java.lang.RunTimeException
It occurs when something went wrong, but our code should be able to neglect or resolve it.
It occurs when something went very wrong and we can not handle it

Throwing Exception

- Sometimes the developer intends to throw an exception and explicitly inform the user about the error. In those cases, we use a syntax throw exception. E.g.

```java
void fall() throws Exception {
    throw new Exception();
}
```

- There are two types of exceptions in java, checked exceptions and unchecked (runtime) exceptions.

- **checked exceptions must be caught or declared** (the compiler objects if this policy is violated), no corresponding stipulation applies to **unchecked exceptions**.

- You can also define your own exception. Or even extend the Exception class and write it in more detail.

```java
throw new Exception();
throw new Exception("an exception.");
throw new RuntimeException();
throw new RuntimeException("a run time error.”);
```

Try/Catch Block

- Exception handling will be done in Java through a try/catch + finally block.

```
try {
        // Protected code block
       //  will be written here
    }
catch (FileIOException e1) {
        // Catch block
    }
catch (ClassCastException e2) {
        // Catch block
    }
finally {
        // The finally block always executes.
    }
```

Type of exception we intend to handle

Identifier of the exception

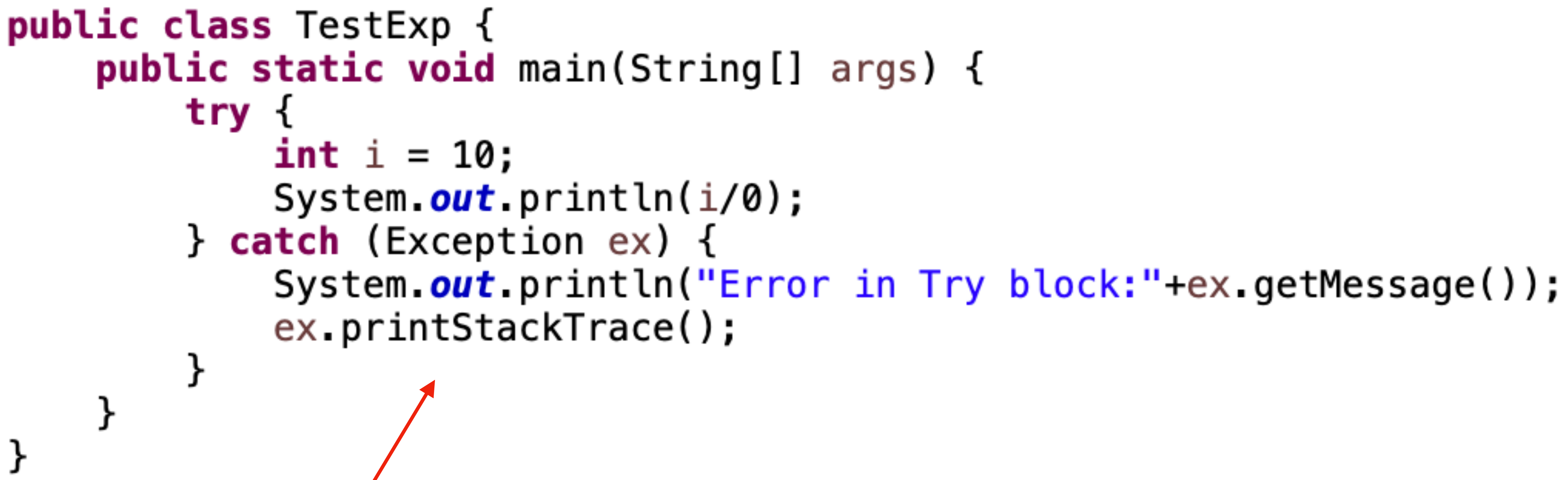
Try/Catch Example

```java
public static void main(String[] args) {
    try {
        int i = 10;
        System.out.println(i/0);
        System.out.println("Here will never reach");
    } catch (Exception ex) {
        System.out.println("Error in Try block:"+ex.getMessage());
        ex.printStackTrace();
    }

}
```

When a statement in a try block causes a run-time problem (throws an exception), the **remaining statements in the try block are skipped and control goes to one of the catch blocks** (the matching catch block).

Try/Catch Example



```
public class TestExp {
    public static void main(String[] args) {
        try {
            int i = 10;
            System.out.println(i/0);
        } catch (Exception ex) {
            System.out.println("Error in Try block:"+ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

I recommend always use it.
It is very useful, while you are in debugging mode

Try/Catch Example

There are three ways to print an exception in Java:

```
System.out.println(e);

System.out.println(e.getMessage());

e.printStackTrace();
```

```
public class TestExp {
    public static void main(String[] args) {
        try {
            int i = 10;
            System.out.println(i/0);
        } catch (Exception ex) {
            System.out.println("Error in Try block:"+ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

I recommend always use it.
It is very useful.

Another Example

```java
try {
    throw new Exception();
    }
catch (Exception ex) {
    System.out.println("exception caught and processed        ");
    }
```

Does it work ?

Another Example

```
try {
    throw new Exception();
    }
catch (Exception ex) {
    System.out.println("exception caught and processed        ");
    }
```

Does it work ?

YES

When to Throw Exception

- We can throw exception inside the try block.

- We can throw exception at the method declaration. e.g.

```
public void test() throws Exception {
```

- When we are calling a method that throws an exception, the rules are the same as within a method, we must handle the exception in Try/Catch block.

- It is also possible to have another try/catch inside the catch block. It is rarely used, but we should know that it is possible.

Some well known Java Exceptions

- NullPointerException
- ClassNotFoundException
- ArrayIndexOutofBoundException
- ClassCastException
- IOException
- IllegalArgumentException
- NumberFormatException
- FileNotFoundException

Some well known Errors

Errors are thrown by JVM. They are rare but they can happen. <u>They can not be handled or declared</u>, just we need to learn why do they occur.

`ExceptionInInitializerError:` Java runs `static` initializers the first time a class is used. If one of the `static` initializers throws an exception, Java can't start using the class.

`StackOverflowError:` When Java calls methods, it puts parameters and local variables on the stack. After doing this a very large number of times, the stack runs out of room and overflows.

`NoClassDefFoundError:` When a method calls an object or a class that does not existed, this error will be raised.

`OutOfMemoryError:` When there is no enough space in the memory available.

Some notes on managing Exception

- `Exception` class will handle all types of exceptions, but it is better to be more specific about the exception you intend to catch in your exception block.

- If you intend to use `Exception` as well as its other subclasses, always remember that `Exception` should be in the last catch.

- Exception checking too liberally is not a good design attitude. Therefore, try to be specific with your exceptions as much as possible. However, some times it is not feasible and we should be liberal with exceptions, , i.e. `Catch (Exception…`

A common mistakes, especially when you are moving from Python to Java

```
try  // DOES NOT COMPILE
  fall();
catch (Exception e)
  System.out.println("get up");
```

The problem is that the braces are missing. It needs to look like this:

```
try {
  fall();
} catch (Exception e) {
  System.out.println("get up");
}
```

Files and I/O

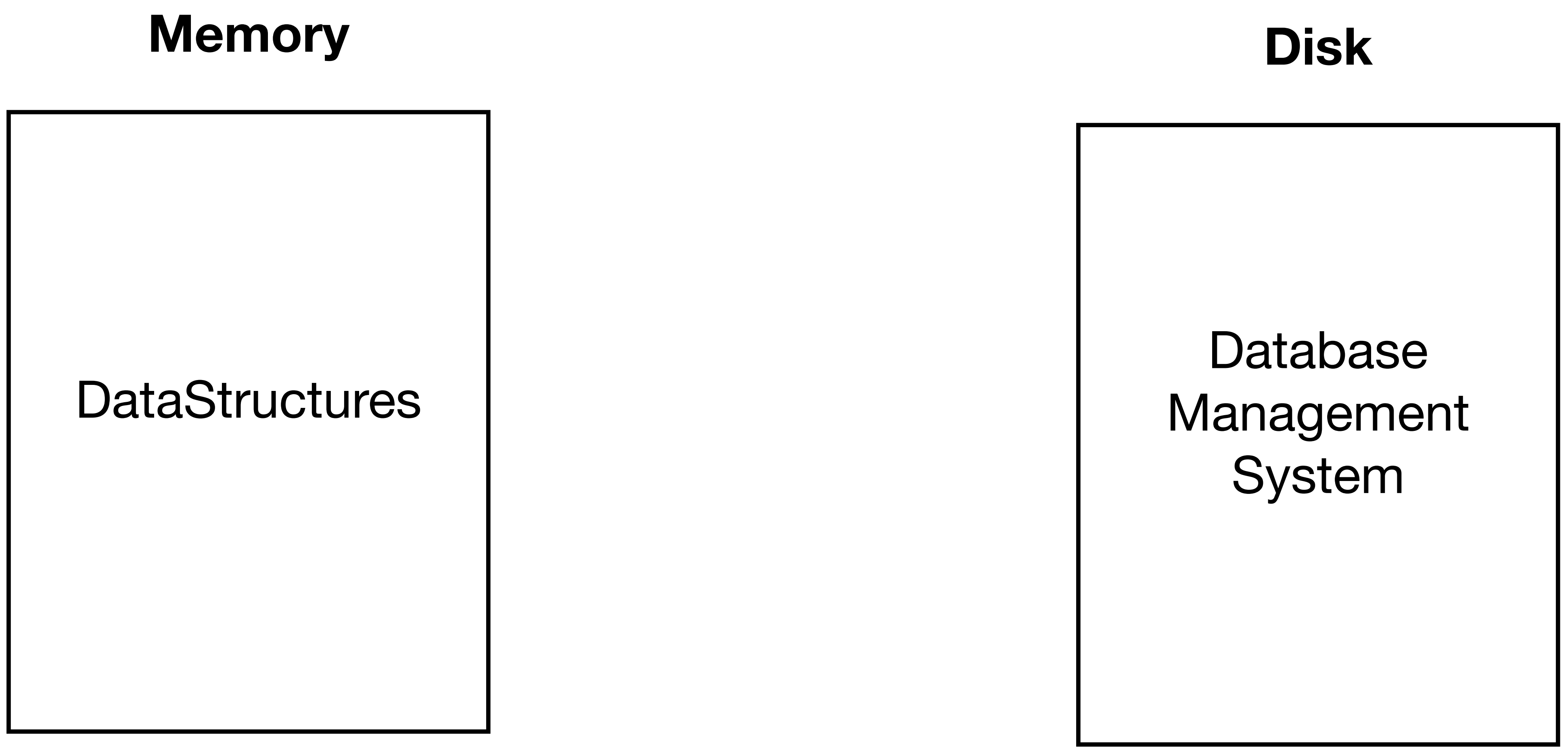
Why we store data on file

- Computer memories are limited and they are very expensive to store the data. Therefore, second storage which is disk, is used as a permanent storage of the data.

- Nevertheless, a good design creates a balance between what to keep in memory and what to keep on disk.

- Memory is associated with fast access, small size and temporary storage. Disk is associated with slow access, large size and permanent storage.

Memory vs Disk



File Example
(do it in the class)

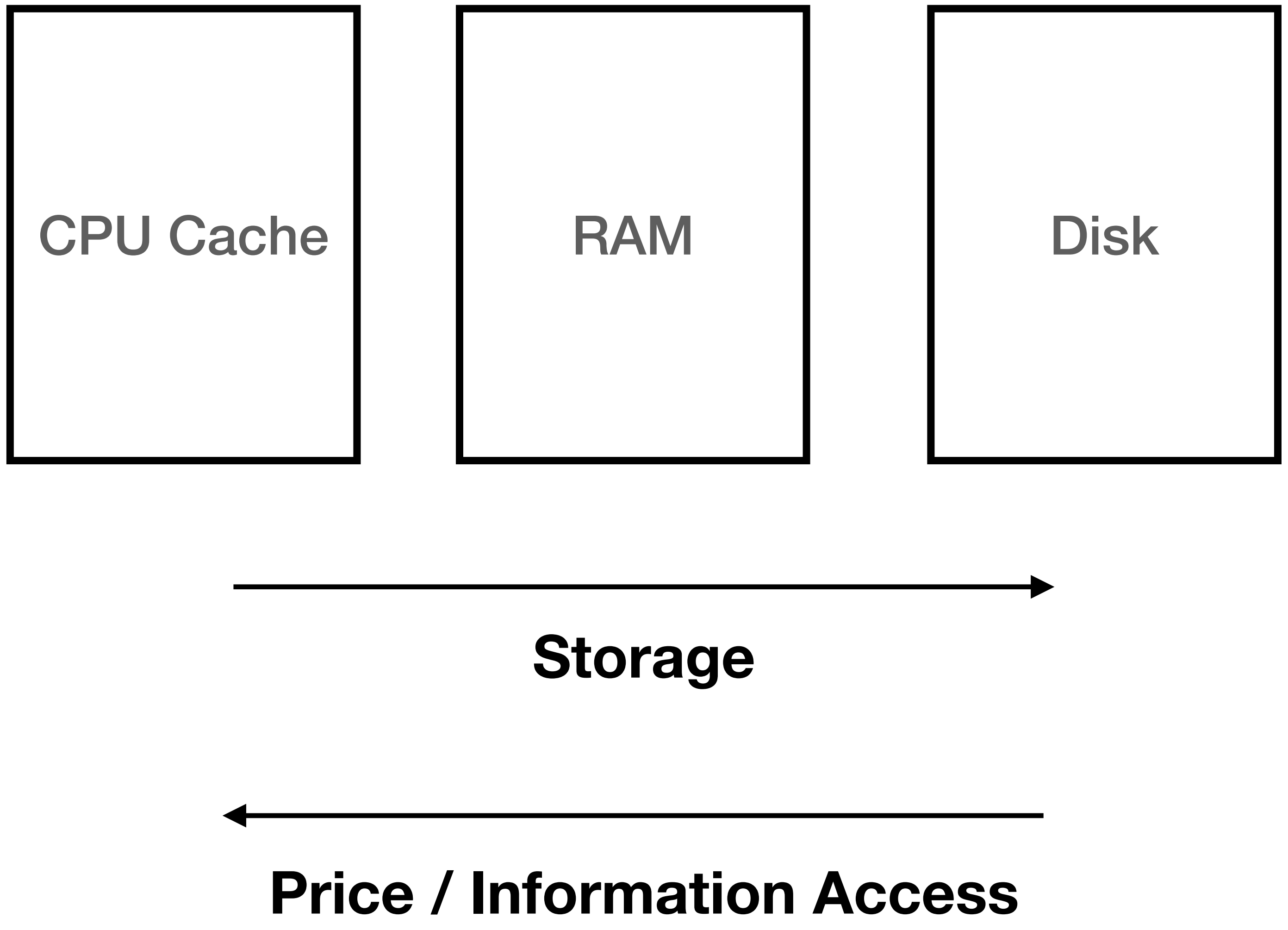
```
import java.io.File;
public class ReadFileInformation {
 public static void main(String[] args) {
  File file = new File(“C:\\data\\ttttest.txt”);
  System.out.println("File Exists: “+file.exists());
  if(file.exists()) {
    System.out.println("Absolute Path: “+file.getAbsolutePath());
    System.out.println("Is Directory: “+file.isDirectory());
    System.out.println("Parent Path: "+file.getParent());
    if(file.isFile()) {
      System.out.println("File size: "+file.length());
      System.out.println("File LastModified: "+file.lastModified()); }
    else {
      System.out.println(“ File does not exist”);
    } // 2nd if
   } // 1st if
 } // main
} // class
```

Why File and Databases?

- So far, we have learned that we can keep data in a file.
- Java Collections are very useful, but they keep the data in the memory and not on the disk.
- Memory is a temporary storage for data, and after the power shuts off or is disconnected, the content will fade away.
- Therefore, we need a long-term place to keep our data. In this scenario, we can store data in a file.

Why File and Databases?

- Files are very useful for the permanent storage of data, and they keep the data on the disk.
- However, when the data gets large, it is a cumbersome process to find information from a file.
- In particular, querying information from raw files is not efficient. Not efficient here means not resource efficient (time, energy,…)
- To mitigate this challenge, databases have been introduced.


CPU Cache
RAM
Disk
Storage
Price / Information Access

CPU Cache

- CPU caches are small pools of memory that store information the CPU is most likely to need next.

- The goal of the cache system is to ensure that the CPU has the next bit of data it will need already loaded into cache by the time it goes looking for it (also called a <u>cache hit</u>). A <u>cache miss</u>, means the CPU has to find the data elsewhere.

- Every CPU core, even ultra-low power ones to the highest-end ones, e.g., Intel Core i9 uses caches. Microcontroller has also a cache.

Source: https://www.extremetech.com/extreme/188776-how-l1-and-l2-cpu-caches-work-and-why-theyre-an-essential-part-of-modern-chips

CPU cache size

- L1 cache access time is almost immediate (< 1 ns), but the quantity is extremely limited (e.g. 32 KB). L1 cache typically has a hit rate between 95 and 97 percent.
- L2 cache access time is pretty low (7 ns), and the quantity is significantly larger (e.g. 256 KB).
- L3 cache access time is noticeably larger (25 ns), but the size is so large that the analogy falls apart (e.g. 8 MB).
- Main memory access time is huge (100 ns), but the capacity is significantly larger than CPU caches (e.g. 16 GB).

Source: https://blogs.oracle.com/javamagazine/post/java-and-the-modern-cpu-part-1-memory-and-the-cache-hierarchy

CPU Cache

- When a CPU can not find information in L1 cache, it searches in the L2 cache while it’s slower, it’s also much larger. If the L2 cache misses as well, it searches L3 and then if existed L4. If not it goes and tries to find the data in main memory, a.k.a. DRAM.
- Frequently missing cache has a negative impact on the CPU performance.
- Some processor share data in caches and some do not share the data in caches.

CPU Cache and Java

- Java does not provide methods to read or write into CPU caches.

Files in Java

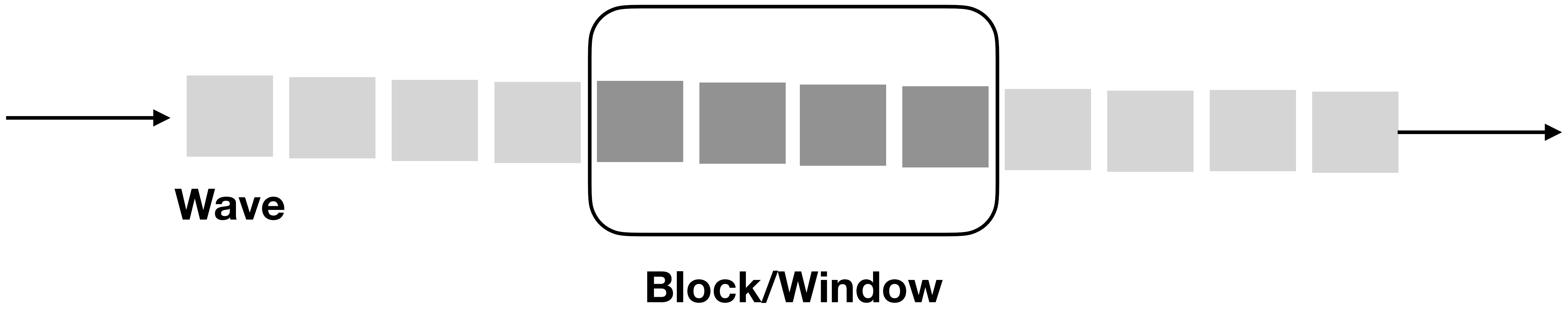
- Java provides lots of useful libraries to store data in files and work with files.
- Java treats a file as a **sequential stream of bytes**.
- Every operating system has a mechanism to mark the **end-of-file**. A Java program starts reading the file until it encounters the marker of the end-of-file.
- There are mainly two types of files in Java: **Text files** and **binary files** (image, audio, video).

File Object Methods

- `exists()`
- `getName()`
- `getAbsolutePath()`
- `isDirectory()`
- `isFile()`
- `length()`
- `lastModified()`
- `delete()`
- `renameTo(File)`
- `mkdir()`
- `getParent()`
- `listFiles()`

Stream

- Stream is a never ending dataset.
- For example, running application log files, online financial transactions, sensor data,…
- Stream processing allow application to access only small portion of data.



`java.io`

- `java.io` defines two set of classes to work with file.

  1. Classes with `Stream` in their name, e.g. `FileInputStream`.

  2. Classes with `Reader/Writer` in their name, e.g. `FileReader`. They are still dealing with streams but the name stream is not mentioned in their name.

- Stream classes are used to read/write any binary files, (e.g. .png, .avi, .mp3) but reader/writer classes are used to read and write character files (e.g. .txt)

`java.io`

- For most input streams there is a similar output stream existed, e.g. `FileInputStream` and `FileOutputStream`.
- Also the most reader classes have corresponding writer classes as well. e.g. `FileWriter` and `FileReader`. But PrintWriter has no PrintReader.
- Streams are categorized into low-level streams and high level streams.

Low level and High level Streams

- Low level streams connects directly to the file on the disk.

- A *high-level stream* is built on top of another stream using wrapping. Wrapping is the process that passes an instance of a class to a constructor of another class.

high level

```
try {
  BufferedReader bufferedReader = new BufferedReader(
    new FileReader(“test.txt"))) {
      System.out.println(bufferedReader.readLine());
}
```

low level

Buffered Reader

- Buffered classes are reading or writing data in a chunk and not byte per byte or character per character. Therefore, disk access operations are getting less and this leads to a significant performance improvement.

- Do not forget to use Buffered reader, especially in your coding exams. Exam takers are usually sensitive about this.

BaseClasses

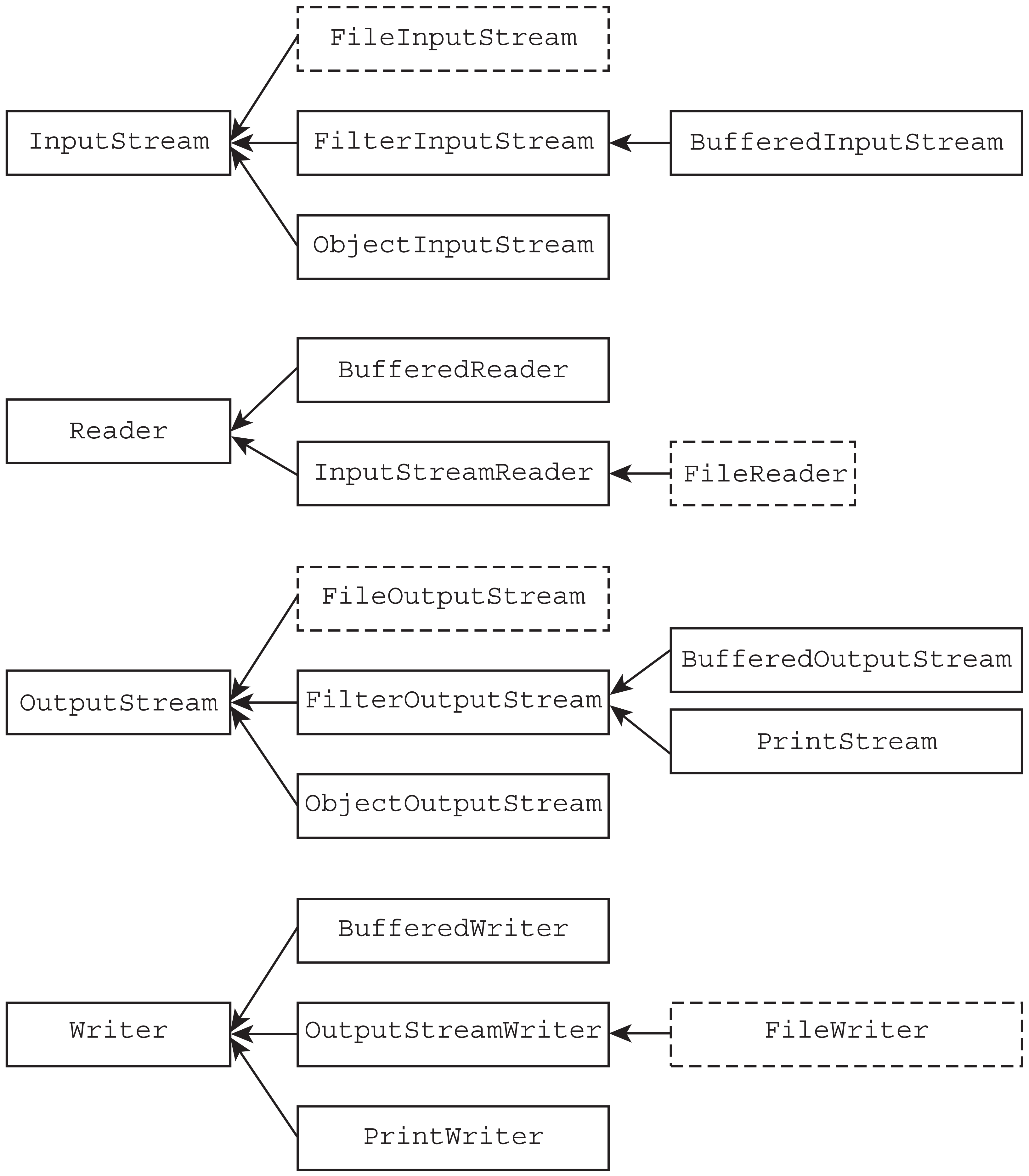
- `java.io` defines four bases classes that constitute the foundation of java file operations, including: `InputStream`, `OutputStream`, `Reader` and `Writer`.
- Children classes of these classes include their name, e.g. `ObjectOutputStream, FileReader, …`

Review of `java.io` Class Properties

- A class with the word <u>InputStream</u> or <u>OutputStream</u> in its name is used for *reading or writing binary data*, respectively.
- A class with the word <u>Reader</u> or <u>Writer</u> in its name is used for reading or writing *character or string data*, respectively.
- Most, but not all, input classes have a corresponding output class.
- A low-level stream connects directly with the source of the data.
- A high-level stream is built on top of another stream using wrapping.
- A class with <u>*Buffered*</u> in its name reads or writes data in groups of bytes or characters and often <u>*improves performance*</u> in sequential file systems.

`java.io` classes

- `InputStream`
  - `FileInputStream`
  - `ObjectInputStream`
  - `InputStreamReader`
- `OutputStream`
  - `FileOutputStream`
  - `ObjectOutputStream`
  - `OutputStreamWriter`
  - `PrintStream`
  - `PrintWriter`
- `Reader`
  - `FileReader`
  - `BufferedReader`
- `Writer`
  - `FileWriter`
  - `BufferedWriter`


InputStream
FileInputStream
FilterInputStream
BufferedInputStream
ObjectInputStream
Reader
BufferedReader
InputStreamReader
FileReader
OutputStream
FileOutputStream
FilterOutputStream
BufferedOutputStream
PrintStream
ObjectOutputStream
Writer
BufferedWriter
OutputStreamWriter
FileWriter
PrintWriter

‘Not’ Compiling Examples

```
new BufferedInputStream(new FileReader("test.txt"));

new BufferedWriter(new FileOutputStream("ttt.png"));

new ObjectInputStream(new FileOutputStream(“ttt.txt"));

new BufferedInputStream(new InputStream());
```

Serialization and Deserialization

- The process of **converting an in-memory object to a stored data format is referred to as *serialization***, with the reciprocal process of **converting stored data into an object, which is known as *deserialization***.

- To make an object serializable that class should implement `java.io.Serializable`.

- Many of the java object we use like String, implements serializable.

- Note in a hierarchy of objects all of them must be marked serializable. For example, if we mark *student* object serializable its *courses* which is another object inside the student object should be serializable as well.

Making an Example Class Serializable

```
import java.io.Serializable;
public class Animal implements Serializable {
 private static final long serialVersionUID = 1L;
 private String name;
 private int age;
 private char type;
 public Animal(String name, int age, char type) {
  this.name = name;
  this.age = age;
  this.type = type;
 }
 public String getName() { return name; }
 public int getAge()  { return age; }
 public char getType() { return type; }
 public String toString() {
   return "Animal [name=" + name + ", age=" + age + ", type=" + type + "]";
 }}
```

```
FileOutputStream fout = new
FileOutputStream(“..path..”);
ObjectOutputStream oos =
newObjectOutputStream(fout);
oos.writeObject(Animal);
```

Making an Example Class Serializable

```
import java.io.Serializable;
public class Animal implements Serializable {
 private static final  long serialVersionUID = 1L;
 private String name;
 private int age;
 private char type;
 public Animal(Stri
  this.name = name;
  this.age = age;
  this.type = type;
 }
 public String getN
 public int getAge()  { return age; }
 public char getType() { return type; }
 public String toString() {
   return "Animal [name=" + name + ", age=" + age + ", type=" + type + "]";
 }}
```

It is not mandatory to have serialVersionUID, but it is recommended to do so.
Otherwise, java is going to create it for us.

Java I/O Example 1
Creating a file

```
import java.io.*;
public class FileExample {
    public static void main(String[] args) {
        try {
            File file = new File("javaFile123.txt");
            if (file.createNewFile()) {
                System.out.println("New File is created!");
            } else {
                System.out.println("File already exists.");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Java I/O Example 2
List content of a folder

```
import java.io.*;
public class FileExample {
  public static void main(String[] args) {
    File f = new File(“/Users/reza/tessst“);
    String filenames[] = f.list();
    for(String filename:filenames){
        System.out.println(filename);
    }
  }
}
```

Java I/O Example 3
Delete a file

```
import java.io.File;

public class MyMain {
   public static void main(String[] args) throws Exception {
      deleteDir(new File(“/home/xxxxx“));
   }
   public static boolean deleteDir(File dir) {
      if (dir.isDirectory()) {
         String[] children = dir.list();
         for (int i = 0; i < children.length; i++) {
            boolean success = deleteDir (new File(dir, children[i]));

            if (!success) {
               return false;
            }
         }
      }
      return dir.delete();
      // System.out.println(“The directory is deleted.");
   }
```

Java I/O Example 4
Read the content of a text file

```
package edu.met622.testio;
import java.io.*;
public class MyFileReader {
    public static void main(String[] args) {
        String testFile = “/Users/home/test/test.txt”;
        BufferedReader br = null;
        String line = "";
        try {
            br = new BufferedReader(new FileReader(testFile));
            while ((line = br.readLine()) != null) {
                System.out.println(“read:” +line);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }}}
```

Java I/O Example 5
Copy binary Files

```
import java.io.*;
public class CopyBufferFileSample {
  public static void copy(File source, File dest) throws  IOException {
  try {
          FileInputStream fin = new FileInputStream(source);
          FileOutputStream fout = new FileOutputStream(dest);
          byte[] b = new byte[1024];
          int noOfBytes = 0;
          System.out.println("Copying file using streams");
             while( (noOfBytes = fin.read(b)) != -1 ){
             fout.write(b, 0, noOfBytes);
          }
          System.out.println("File copied!");
          //close both streams
          fin.close();
          fout.close();
       }
       catch(FileNotFoundException fnf){
          System.out.println("Specified file not found :" + fnf);
       }
       catch(IOException ioe){
          System.out.println("Error while copying file :" + ioe);
       }
    }
}
```

Example 6
Read from file

- Create a mock text (.txt) file with code (not by hand).
- Write inside that file for 30 times following sentences:

  I am expert in working with java I/O.

Copy a text file

```
import java.io.*; import java.util.*;
public class CopyTextFileSample {
public static List<String> readFile(File source) throws IOException {
List<String> data = new ArrayList<String>();
try (BufferedReader reader = new BufferedReader(new FileReader(source))) {
String s;
while((s = reader.readLine()) != null) {
data.add(s); }
}
return data; }
```

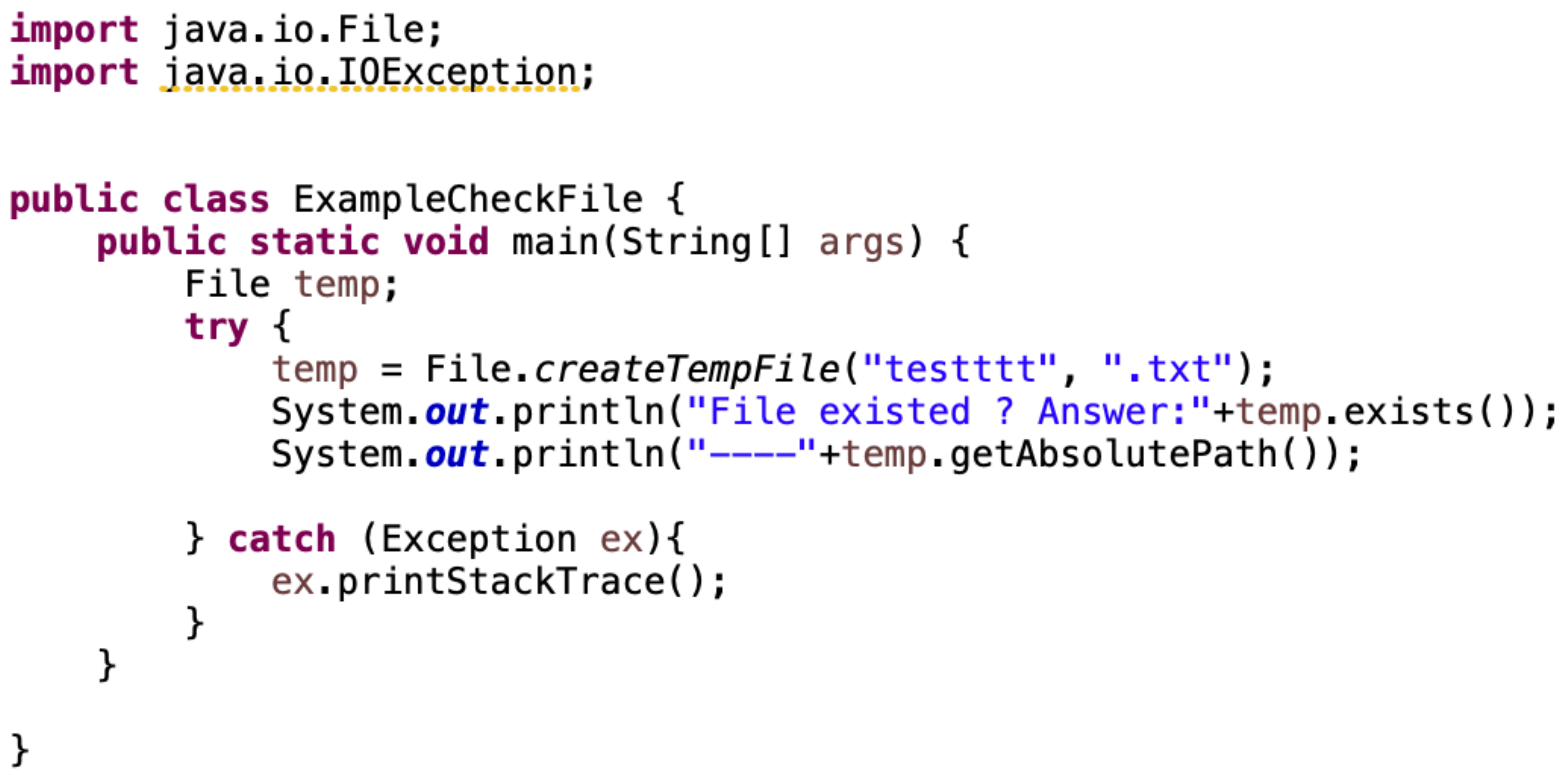
```
public static void writeFile(List<String> data, File destination) throws IOException
{
try (BufferedWriter writer = new BufferedWriter(
new FileWriter(destination))) {
for(String s: data) { writer.write(s);
writer.newLine(); }
} }
```

```
public static void main(String[] args) throws IOException { File source = new
File("Zoo.csv");
File destination = new File("ZooCopy.csv");
List<String> data = readFile(source);
for(String record: data) { System.out.println(record);
}
writeFile(data,destination); }
}
```

Absolute vs Relative Path

- A relative path is a path relative to another directory.
- An absolute path is a path that we use to access the file in the operating system.
- We can also get the path from URI (Uniform Resource Indicator), if the file is on the web. E.g. URI in windows platform: `file:\\C:\example.txt`
  Example of URI in Unix systems:
  `file:/home/username/test.txt`

Example of Checking if a temp file existed.



```java
import java.io.File;
import java.io.IOException;


public class ExampleCheckFile {
    public static void main(String[] args) {
        File temp;
        try {
            temp = File.createTempFile("testttt", ".txt");
            System.out.println("File existed ? Answer:"+temp.exists());
            System.out.println("----"+temp.getAbsolutePath());

        } catch (Exception ex){
            ex.printStackTrace();
        }
    }

}
```

Example of Checking if a temp file existed.

```java
import java.io.File;
import java.io.IOException;


public class ExampleCheckFile {
    public static void main(String[] args) {
        File temp;
        try {
            temp = File.createTempFile("testttt", ".txt");
            System.out.println("File existed ? Answer:"+temp.exists());
            System.out.println("----"+temp.getAbsolutePath());

        } catch (Exception ex){
            ex.printStackTrace();
        }
    }

}
```

```
File existed ? Answer:true
----/var/folders/jr/tpgq1lds0rx3dd59myq8rn7h0000gn/
T/testttt3491889226670552867.txt
```

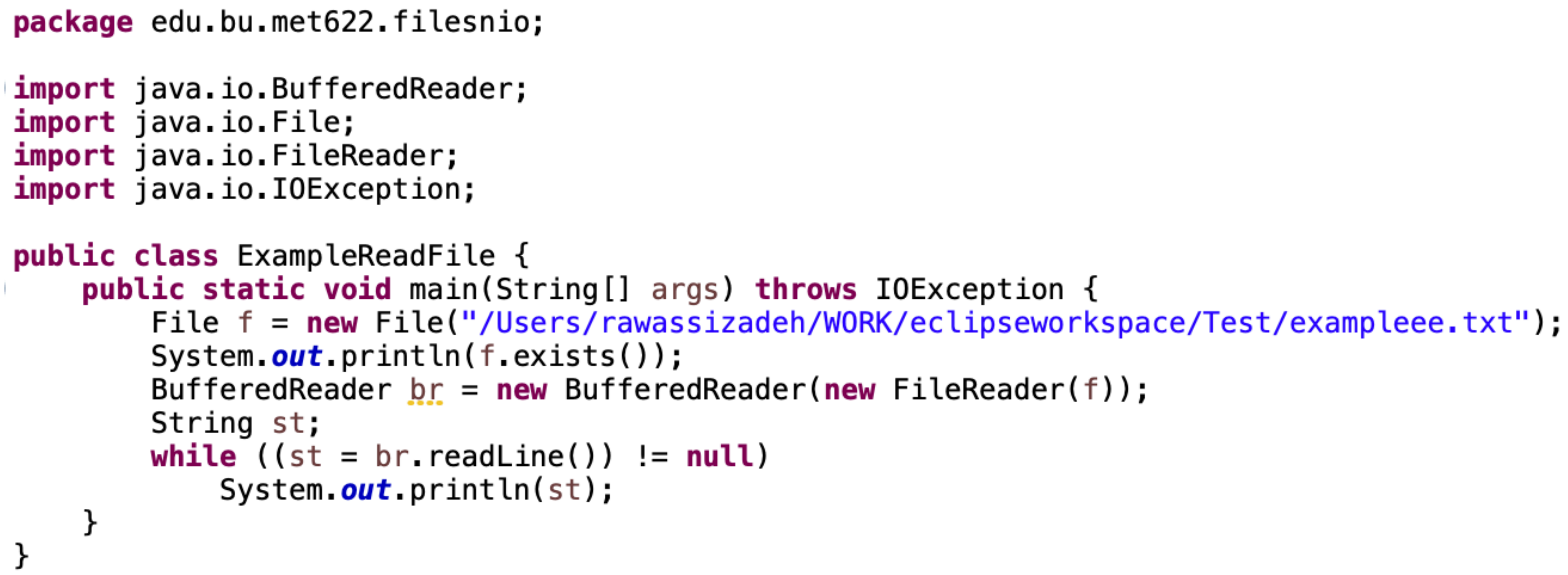
Example to create a text file and write in it.

```java
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class ExampleWriteinTextFile {

    public static void main(String[] args) throws Exception {
        String text = "Hello world";
        BufferedWriter output = null;
        try {
            File file = new File("exampleee.txt");
            output = new BufferedWriter(new FileWriter(file));
            output.write(text);
        } catch ( IOException e ) {
            e.printStackTrace();
        } finally {
          if ( output != null ) {
            output.close();
          }
        }
    }
}
```

Example to read a content of a text file.



```java
package edu.bu.met622.filesnio;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class ExampleReadFile {
    public static void main(String[] args) throws IOException {
        File f = new File("/Users/rawassizadeh/WORK/eclipseworkspace/Test/exampleee.txt");
        System.out.println(f.exists());
        BufferedReader br = new BufferedReader(new FileReader(f));
        String st;
        while ((st = br.readLine()) != null)
            System.out.println(st);
    }
}
```

Example to read a content of a text file.

separator

in windows: \

in Unix based OS: /

```
package edu.bu.met6

import java.io.Buff
import java.io.File
import java.io.File
import java.io.IOEx

public class ExampleReadFile {
    public static void main(String[] args) throws IOException {
        File f = new File("/Users/rawassizadeh/WORK/eclipseworkspace/Test/exampleee.txt");
        System.out.println(f.exists());
        BufferedReader br = new BufferedReader(new FileReader(f));
        String st;
        while ((st = br.readLine()) != null)
            System.out.println(st);
    }
}
```

Example to read a content of a text file.

```java
package edu.bu.met622.filesnio;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class ExampleReadFile {
    public static void main(String[] args) throws IOException {
        File f = new File("/Users/rawassizadeh/WORK/eclipseworkspace/Test/exampleee.txt");
        System.out.println(f.exists());
        BufferedReader br = new BufferedReader(new FileReader(f));
        String st;
        while ((st = br.readLine()) != null)
            System.out.println(st);
    }
}
```

```
true
Hello world
Hello again
Helllllooooo world
```

Regular Expression

Some general note about Regular Expression

- The regular expression **"."** (a single period) represents (matches) any single character. E.g., the string "Book" matches all of the following regular expressions: "B..." "Bo.k" "B.ok" "Boo." ".ook" "...." "B..k" .

- The regular expression **"[abc]"** means one of the three characters, that is, a or b or c. For example, the strings "bxb", "bxc", "bxa" all match the regular expression “bx[abc]".

- The regular expression **"(nd|m)"** represents the <u>two characters nd (in that order) or the single character m</u>. For example, both "And" and "Am" match the regular expression "A(nd|m)".

Some general note about Regular Expression

- In many languages, e.g. shell scripting and SQL the regular expression **“?"** represents any single character.

- In many languages, **“ * ”** (a single star) presents (matches) any number of possible characters.

Java Regular Expressions Methods

- `boolean matches()` whether the regular expression matches the pattern.
- `boolean find()` finds the next expression that matches the pattern.
- `boolean find(int start)` finds the next expression that matches the pattern from the given start number.
- `String group()` returns the matched subsequence.
- `int start()` returns the starting index of the matched subsequence.
- `int end()` returns the ending index of the matched subsequence.
- `int groupCount()` returns the total number of the matched subsequence.

https://www.javatpoint.com/java-regex

Java Example with Regular Expression

```
package edu.bu.met622.filesnio;
import java.util.regex.*;
public class RegexExample {
    public static void main(String[] args) {

        String txt= "This is a sample text used for Java regular expression.";

        // Create a Pattern object
        //Pattern p = Pattern.compile(".ext");
        //Pattern p = Pattern.compile("fo*");
        Pattern p = Pattern.compile("\\b");

        // Get a matcher object
        Matcher m = p.matcher(txt);

        int count = 0;

        while(m.find()) {
             count++;
             System.out.println("Match number "+count);
             System.out.println("start(): "+m.start());
             System.out.println("end(): "+m.end());
        }
    }
}
```

StringBuilder

```
package edu.bu.met622;

public class Filexx {
     public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Hello");
        System.out.println(sb);

        sb.append(" My ");
        sb.append("Friend!");
        System.out.println(sb);

        sb.insert(9, "New "); System.out.println(sb);

        sb.delete(6, 9); System.out.println(sb);

        sb.deleteCharAt(11); System.out.println(sb);

        sb.replace(10, 15, “Neighbor"); System.out.println(sb);

        sb.reverse();  System.out.println(sb);
     }
  }
```

StringBuilder

```
public class Filexx {
 public static void main(String[] args) {
  StringBuilder sb = new StringBuilder("Hello"); System.out.println(sb);
  sb.append(" My "); sb.append("Friend!"); System.out.println(sb);
  sb.insert(9, "New "); System.out.println(sb);
  sb.delete(6, 9); System.out.println(sb);
  sb.deleteCharAt(11); System.out.println(sb);
  sb.replace(10, 15, "Neighbor"); System.out.println(sb);
  sb.reverse();
   System.out.println(sb); }
}
```

```
Hello
Hello My Friend!
Hello My New Friend!
Hello New Friend!
Hello New Fiend!
Hello New Neighbor!
!robhgieN weN olleH
```