Exceptions and Errors

Recitation 3

Programming Special Cases

- What does “Hi”.indexOf(‘a’) return?
  - -1
- This method requires checking if the value is -1 when you use it
- Very error prone
  - e.g. String s1 = s2.substring(s2.indexOf("header:") + 7);

Exception vs. Error

- Error
  - Mistake on the part of the programmer
  - Not recoverable (OutOfMemoryError)
- Exception
  - Often used to handle special behaviour in a program (FileNotFoundException)

Checked vs Unchecked Exceptions

Checked:

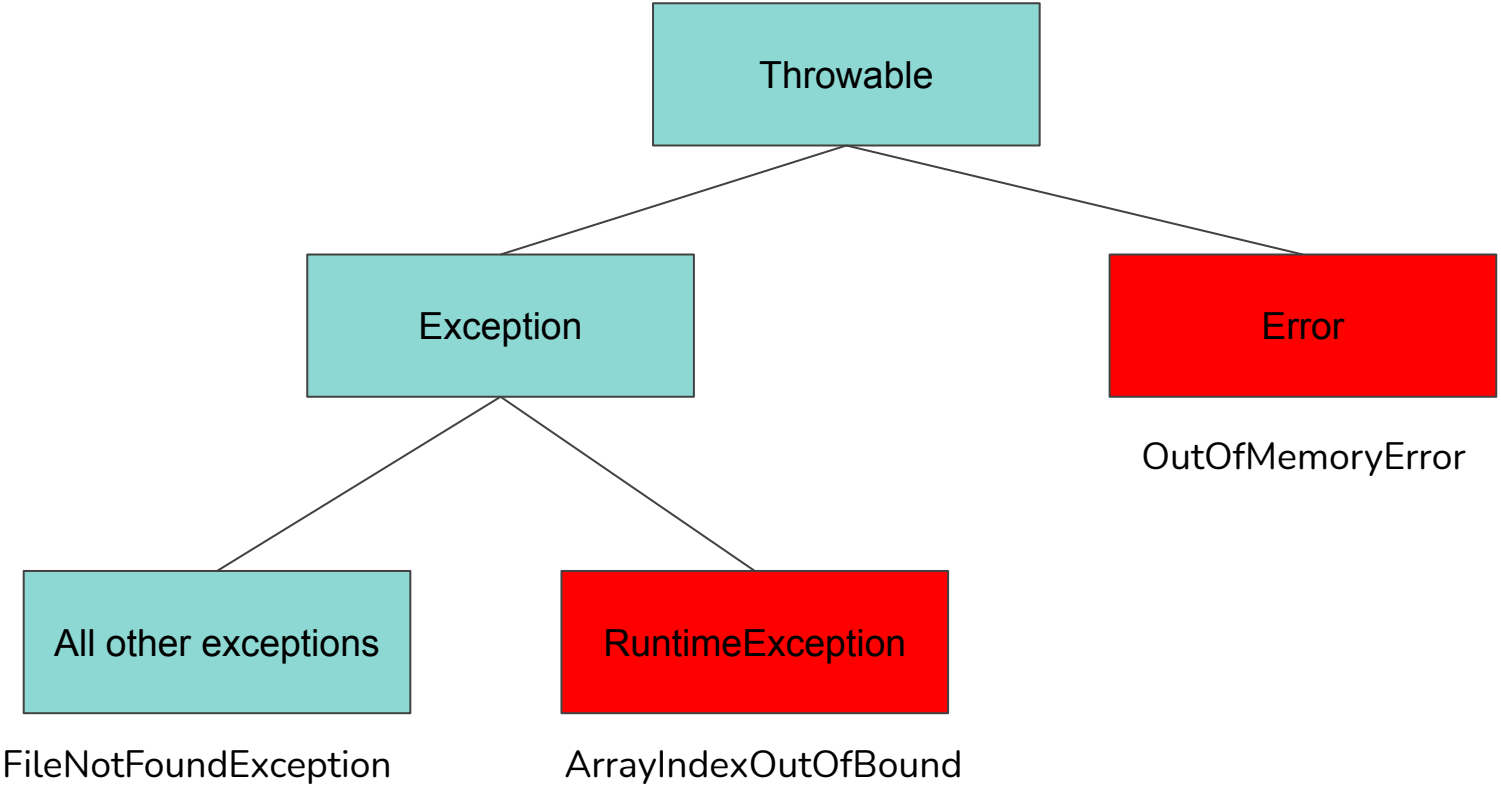
- Required to be handled by either try catch or passing the exception further along as indicated in method header
- Represents unusual but unpreventable circumstances
- Useful to factor out code for rare cases
- Required to be checked
- Ex. IOException

Unchecked:

- Not required to be handled
- Usually represent some kind of programmer error like improper use of array or calling method on null
- Unchecked exceptions are subclass of RuntimeException and Error
- Ex. NullPointerException

Throwable

- If e extends Throwable, throw e is a valid statement
- Exception and Error are subtypes of Throwable
- Exceptions get thrown upward through nested function calls until someone catches it
- The exception is then bound and in scope in the catch block



Red indicates unchecked

Finally Block And Multiple Catches

- Every try block must have at least one catch or finally
- Multiple catches can correspond to different exceptions that may arise in try - catch block (FileNotFoundException, IOException)
- Finally block contains code that gets run regardless of which or if any exceptions are caught
    - Good for essential clean up like closing resources
    - Executes even if method returns in try/catch block

Demo

Exception Syntax

```
1  /**
2           * Constructor: a Room with n rows and m columns of seats.
3           *
4           * @param n Number of rows of seats
5           * @param m Number of columns of seats
6           * @throws IllegalArgumentException if n or m are negative
7           */
8          public Room(int n, int m) {
9                  if (n < 0 || m < 0) {
10                         throw new IllegalArgumentException();
11                 }
12                 people = new String[n][m];
13                 numRows = n;
14                 numCols = m;
15         }
```

Exception Syntax

```
1 public Room makeSquareRoom(int a) {
2     try {
3         return new Room(a,a);
4     } catch (IllegalArgumentException e) {
5         System.err.println("Cannot create negative rows! Returning empty room instead.");
6         return new Room(0,0);
7     }
8 }
```

```java
public Room makeSquareRoom(int a) {
    try {
        return new Room(a,a);
    } catch (IllegalArgumentException e) {
        System.err.println("Cannot create negative rows! Returning empty room instead.");
        return new Room(0,0);
    } finally {
        System.out.println("Room created!");
    }
}
```

Try with Resources

```java
public void printLinesNormalTryFinally(File file) throws IOException {
  BufferedReader br = null;
  try {
    br = new BufferedReader(new FileReader(file));
    // read some stuff from the file
  } finally {
    if (br != null) {
      br.close();
    }
  }
}

public void printLinesTryWithResources(File file) throws IOException {
  try (BufferedReader br = new BufferedReader(new FileReader(file))) {
    // read some stuff from the file
  }
}
```

Specifying Functions with Preconditions

```
/** Returns: length of nth side of a triangle
 *  Requires: 0 <= n <= 2
 */
double sideLength(int n);
```

Can do anything if n is not between 0 and 2.

Function Preconditions: Checks clause

```
/** Returns: length of nth side of a triangle
 *  Checks: 0 <= n <= 2
 */
double sideLength(int n);
```

Better, but what exception will be thrown when the check fails?

Function Preconditions: Assert example

```
/** Returns: length of nth side of a triangle
 *  Checks: 0 <= n <= 2 (assert)
 */
double sideLength(int n) {
    assert n >= 0 && n <= 2;
    ...
}
```

Better, but if an exception is thrown, it still indicates a problem in the client code.

Function Preconditions: Remove Precondition

```
/** Returns: length of nth side of a triangle. Throws
 *           OutOfBoundsException if n < 0 or n > 2.
 */
double sideLength(int n) throws OutOfBoundsException;
```

The exception is now expected behavior in some situations and must be handled by the client.