Software Construction
Exceptions

Dhammika Elkaduwe

Department of Computer Engineering
Faculty of Engineering

University of Peradeniya

ILOs

- What are exceptions
- Exceptions (checked/unchecked) and errors
- File IO (reading/writing files)
- Keywords: try, catch, finally, throws, throw
- FileReader, FileWriter, BufferedReader, BufferedWriter
- Assert: how to define, when to use

What are exceptions?

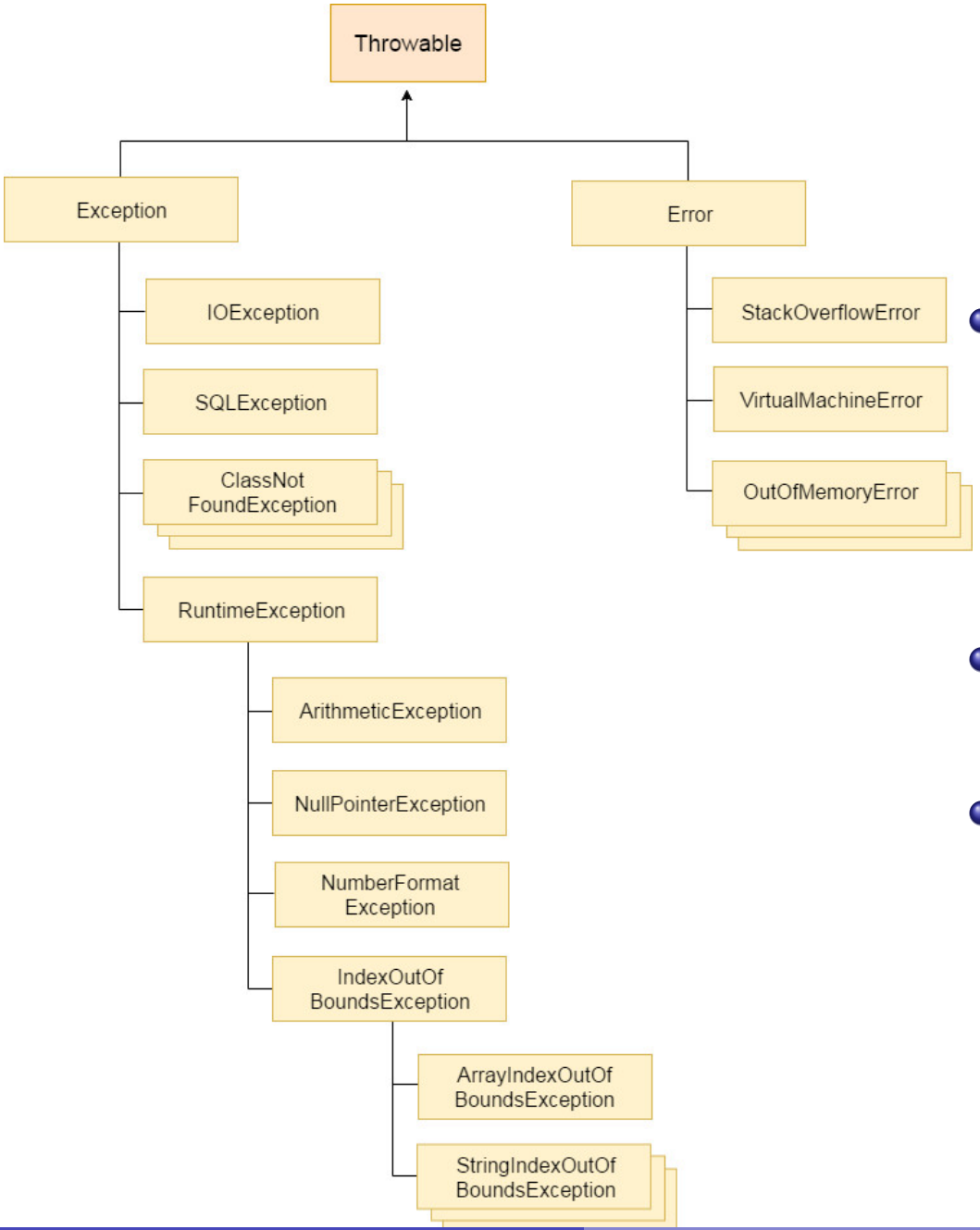
An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

taken from Oracle documents

- Something that should not have happen
- Which disrupts the normal flow
- There are two main types:
  - Exceptions: events from which you **can** recover
  - Errors: events from which one **cannot** recover

```
public void withdraw(float amount) {
  if(getBalance() > amount) balance -= amount;
  // this function should not be called if the balance is less
```

In Java



- Exceptions: two main types
  - Checked Exceptions: These are checked at compile-time
  - Unchecked Exceptions: These are checked at run-time
- Errors: you cannot recover from these
- All are derived from *Throwable* (more on this idea of derived classes next week)

Example: *Cat* Command

Cat: read the content of the given file.

- java cat tx.txt *implies* read and display the content of `tx.txt` on screen
- java cat 1.txt 2.txt $\Longrightarrow$ should display the content of `1.txt` and then `2.txt`
- java cat $\Longrightarrow$ should display error

Cat implementation

```
// display the content of the given file.
public static void show(String fileName) {

  FileReader readDesc = new FileReader(fileName);

  // Buffered readers are faster and works with large files
  // Wrap the readDesc with a buffered reader

  BufferedReader reader = new BufferedReader(readDesc);

  String line = null;

  while((line = reader.readLine()) != null)
     System.out.println(line);

  reader.close();
  readDesc.close();
}
```

File operations

```
  import java.io.*; // for IO operations

  FileReader readDesc = new FileReader(fileName);

  BufferedReader reader = new BufferedReader(readDesc);

  while((line = reader.readLine()) != null)
     System.out.println(line);

  reader.close(); // close file
  readDesc.close();
}
```

see Cat.java

- `import java.io.*` for handling IO activities
- `FileReader` can be used to read the file
- `BufferedReader` makes things faster
- Once done, close the *FileReader* and *BufferedReader*

Exceptions handling: basics

```
  // Following might throw exceptions
  FileReader readDesc = new FileReader(fileName);

  while((line = reader.readLine()) != null)

  reader.close(); // close file
  readDesc.close();
}
```

see Cat.java

Compiling this code will not work!

Why: exceptions are not caught or declared thrown

Exceptions can be:

- Caught (using try, catch and finally keywords)
- Or declare function to throw them

Exceptions handling: try catch

```
  // Following might throw exceptions
  FileReader readDesc = new FileReader(fileName);

  while((line = reader.readLine()) != null)

  reader.close(); // close file
  readDesc.close();
}
```

see Cat.java

Compiling this code will not work!

Why: exceptions are not caught or declared thrown

Exceptions can be:

- Caught (using try, catch and finally keywords)
- Or declare function to throw them

Exceptions handling: try catch blocks

```
try {
  readDesc = new FileReader(fileName);
} catch(FileNotFoundException ex) {
  System.out.println("File " + fileName + " cannot be openned");
}
```

see Cat.java

- Try the code within the *try* block
- Which can throw exceptions
- Catch the exceptions thus thrown using the *catch* block

Exceptions handling: try, catch blocks

```
try {
  while((line = reader.readLine()) != null)
     System.out.println(line);

  reader.close();
  readDesc.close();
} catch(IOException ex) {
  System.out.println("Some IO Exception");
}
```

see Cat.java

- Try the code within the *try* block
- Which can throw exceptions
- Catch the exceptions thus thrown using the *catch* block

Exceptions handling: try, catch blocks

```
try {
  while((line = reader.readLine()) != null)
     System.out.println(line);

  reader.close();
  readDesc.close();
} catch(IOException ex) {
  System.out.println("Some IO Exception");
}
```

see Cat.java

- Try the code within the *try* block
- Which can throw exceptions
- Catch the exceptions thus thrown using the *catch* block

Exceptions handling: try, catch, finally blocks

see Final.java

- If a name is given as command-line argument use it
- If not read "Final.java" file

```
try {
  fileName = args[0];
} catch(IndexOutOfBoundsException ex) {
  fileName = "Final.java"; // default file to read
} finally {
  Cat.show(fileName);
}
```

- Try the code within the *try* block
- If an appropriate exception is thrown catch it within *catch* block
- Execute the *finally* block (in any case)
- (*finally* is used to clean up)
- (you can decide not to catch *IndexOutOfBoundsException* since it is unchecked

Exceptions handling: Catching more than one exception

```
try {
  readDesc = new FileReader(fileName);
  reader = new BufferedReader(readDesc);
  while((line = reader.readLine()) != null)
      System.out.println(line);
  reader.close();
  readDesc.close();
} catch(FileNotFoundException e2) {
  System.out.println("File " + fileName + " cannot be openned");
} catch(IOException e1) {
  System.out.println("Some IO Exception");
}
```

see Cat.java

- You can catch more than exception
- Has to catch more specific exceptions first
  - If you change the order above it will not work
  - *FileNotFoundException* is an *IOException*
  - There are more *IOException*

Exceptions handling: use *throws*

```
public static void show_throws(String fileName)
  throws IOException, FileNotFoundException {
    FileReader readDesc = null;
    BufferedReader reader = null;
    String line = null;
```

see Cat.java

- You are not catching/handling the exceptions
- Declare that they can be thrown
- If one can throw more than one exception, use commas
- If an exception happens it has to caught by the calling function (or declared)

File operations in Java

- Read https://docs.oracle.com/javase/tutorial/essential/io/file.html
- More useful operations

Why use exceptions

- Separate error/exception-handling code from regular code
- Send errors back up the calling stack
- Grouping different errors/exceptions together

Example from the *Account*

see Account.java

```
public void withdraw(float amount)
throws Exception { // declare to throw exceptions
  if(this.balance < amount) throw new Exception();

  this.balance -= amount;
}

// in main
Account a = new Account(34f);
try {
  a.withdraw(123f);
} catch(Exception e) {
  System.out.println("Got an exception");
}
```

- You can create new exceptions by extending the *Exception* class
- More on how to do this with *inheritance*

Assertions

see Account.java

```
public void withdraw_assert(float amount) {
  assert amount < this.balance : "Not enough money";
  this.balance -= amount;
}
```

- Functions have preconditions; conditions on which the function is based on.
- Example: `withdraw` assumes the value to withdraw is less than what is in account. Calling functions need to make sure of this.
- You can use *assert* to enforce this
- Enable assertions with `enableassertions` flag with the JVM
- Without that flag assertions will be disabled
- Ideal for debugging (note the call graph when assertion fail)

```
assert boolean_expression :  ''Error message''
```

ILOs: Revisited

- What are exceptions
- Exceptions (checked/unchecked) and errors
- File IO (reading/writing files)
- Keywords: try, catch, finally, throws, throw
- FileReader, FileWriter, BufferedReader, BufferedWriter
- Assert: how to define, when to use