Exception Handling

saacsos

Exception Handling

- There are two aspects to dealing with program errors: detection and handling
- In Java, exception handling provides a flexible mechanism for passing control from the point of error detection to a handler that can deal with the error.

Throwing Exceptions

Throwing Exceptions

When you detect an error condition. You just throw an appropriate exception object.

```
if (amount > balance)
{
    // Now what?
}
```

throw Statement

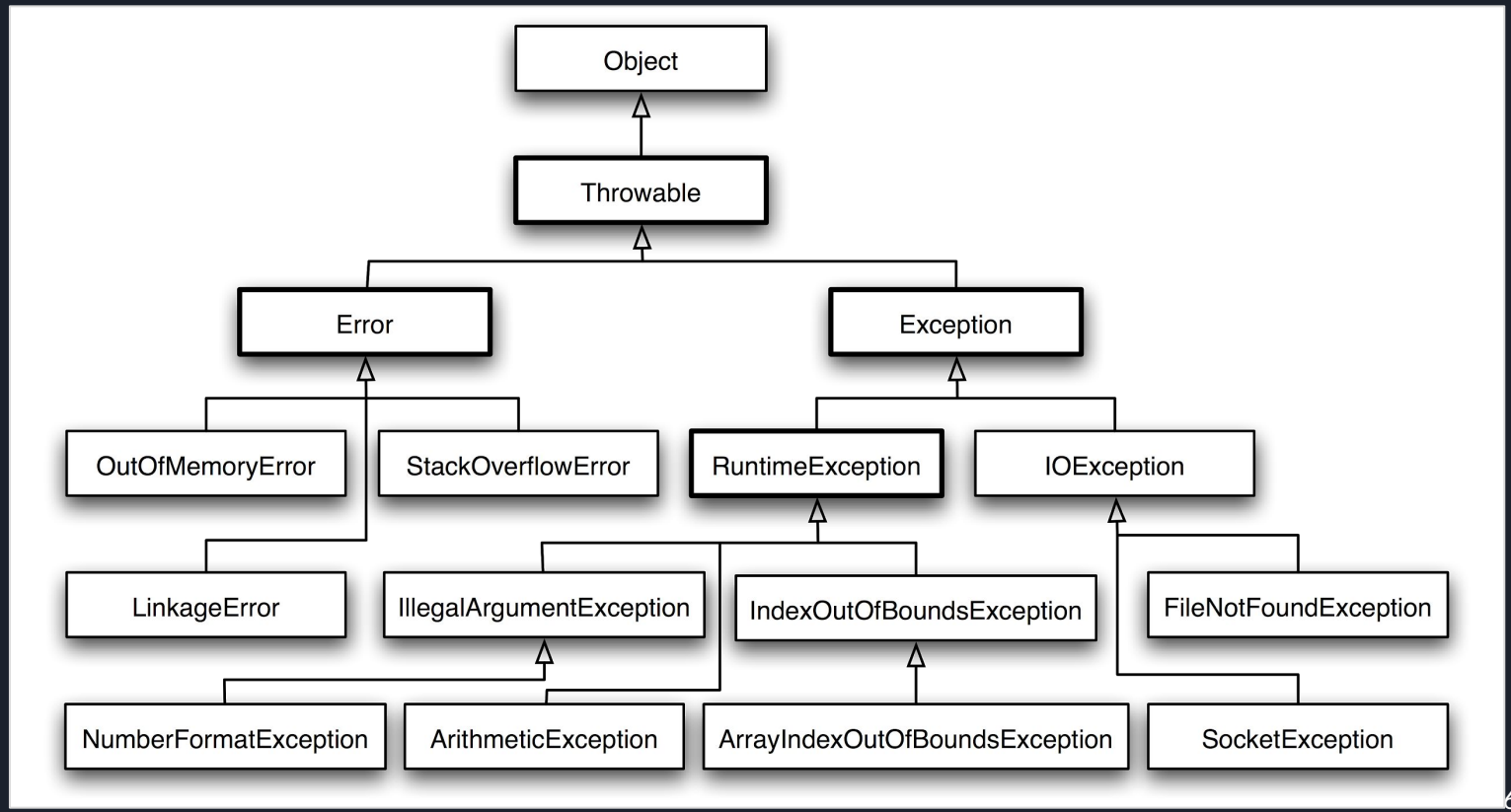
```
if (amount > balance)
{
    throw new IllegalArgumentException(
            "Amount exceeds balance"
    );
}
balance = balance - amount;
```

A new exception object is constructed, then thrown.

Most exception objects can be constructed with an error message.

This line is not executed when the exception is thrown.

Hierarchy of Exception Classes



Catching Exceptions

Catching Exceptions

- Every exception should be handled somewhere in your program.
- If an exception has no handler, an error message is printed, and program terminates.
- Handle exceptions with the try/catch statement.

try/catch Statement

```
try {
   Scanner in = new Scanner(new File("input.txt"));
   String input = in.next();
   process(input);
}
catch (IOException exception){
   System.out.println("Could not open file");
}
catch (Exception except) {
   System.out.println(except.getMessage());
}
```

This constructor can throw a `FileNotFoundException`

This is the exception that was thrown.

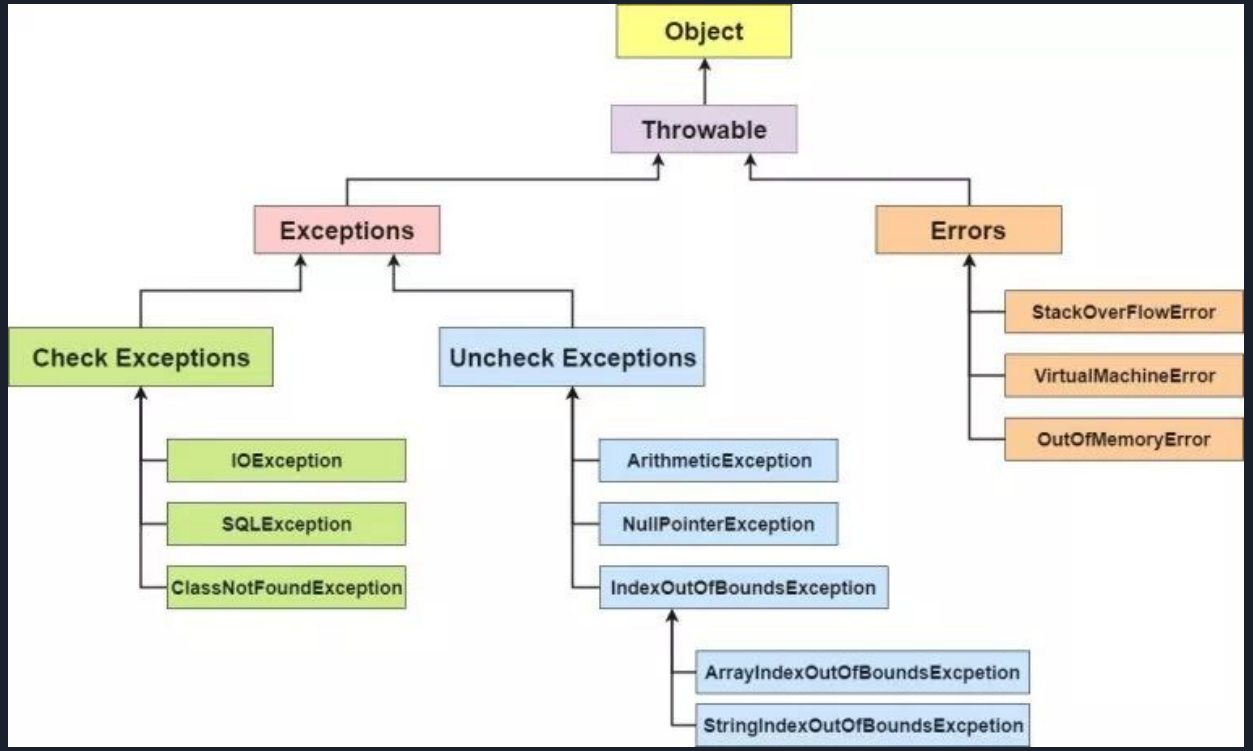
When an `IOException` is thrown, execution resume here.

Additional catch clauses can appear here.

A `FileNotFoundException` is a special case of an `IOException`

Throwable Objects

Throwable Objects



Errors

- Internal errors are reported by subclasses of the type `Error`.
- e.g. `OutOfMemoryError`
- Fatal errors, happen rarely.

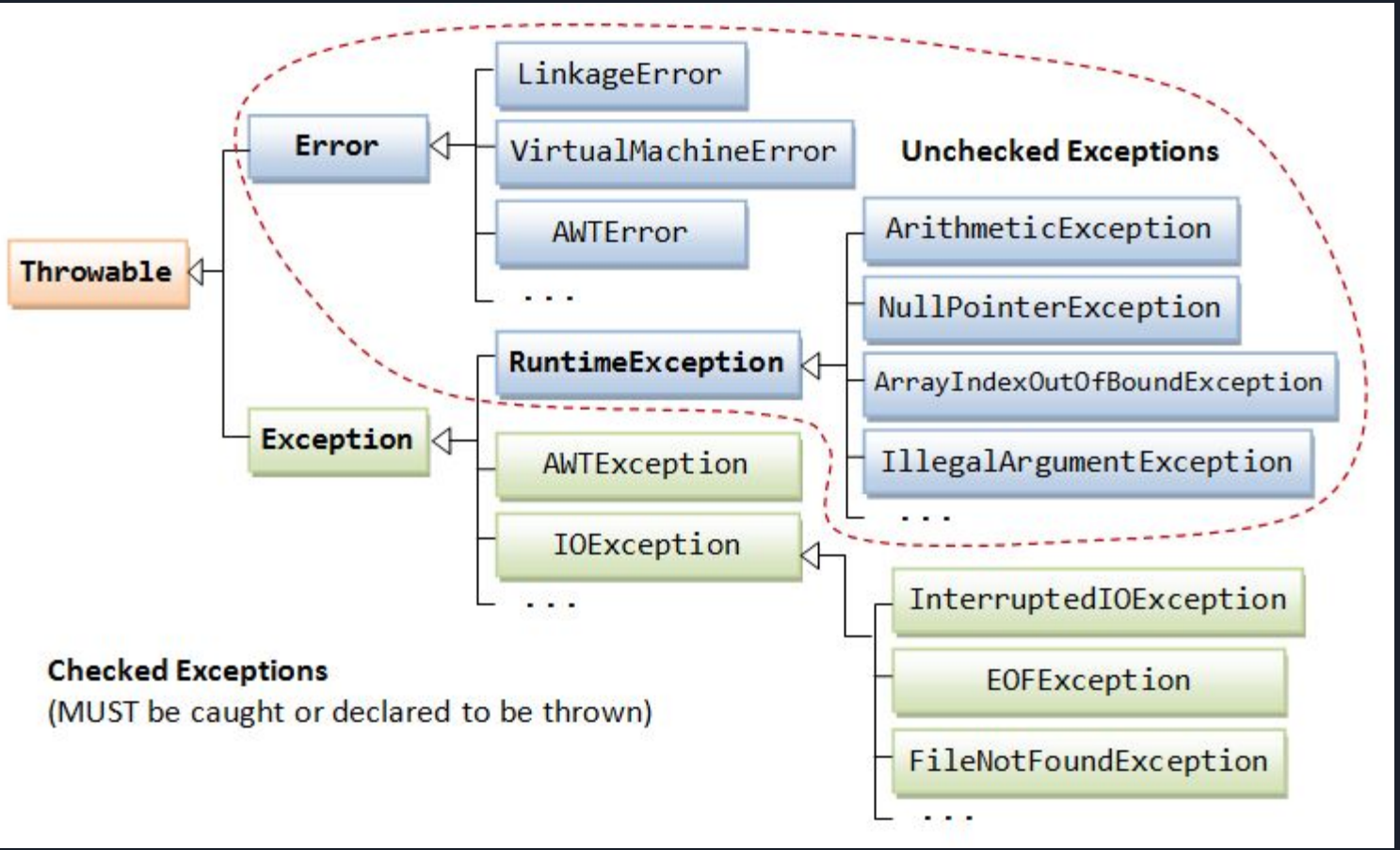
Unchecked Exceptions

- Subclasses of `RuntimeException` are called unchecked exceptions.
- e.g. `IndexOutOfBoundsException`
  `IllegalArgumentException`
  `NullPointerException`
- indicate errors in your code

Checked Exceptions

- All other exceptions are called checked exceptions.
- e.g. `FileNotFoundException`

  `ClassNotFoundException`
- indicate that something has gone wrong for some external reason beyond your control
- MUST be caught or declared to be thrown


Throwable
Error
LinkageError
VirtualMachineError
AWTError
...
Exception
RuntimeException
ArithmeticException
NullPointerException
ArrayIndexOutOfBoundException
IllegalArgumentException
...
AWTException
IOException
...
InterruptedIOException
EOFException
FileNotFoundException
...
Unchecked Exceptions
Checked Exceptions
(MUST be caught or declared to be thrown)

The throws Clause

- The current method cannot handle the exception.
- You need to tell the compiler that you are aware of this exception,
- and that you want your method to be terminated when it occurs.
- You supply the method with a throws clause.

The throws Clause

```
public void readData(String filename)
   throws FileNotFoundException,
          NumberFormatException
{
   File inFile = new File(filename);
   Scanner in = new Scanner(inFile);
   . . .
}
```

- You MUST specify all checked exceptions that this method may throw.
- You may also list unchecked exceptions

The finally Clause

- Once a try block is entered, the statement in a finally clause are guaranteed to be executed.
- Whether or not an exception is thrown.

The finally Clause

```
PrintWriter out = new PrintWriter(filename);
try {
   writeData(out);
}
finally {
   out.close();
}
```

- The variable must be declared outside the try block

  so that the finally clause can access it.

Designing Your Own Exception Types

Designing Your Own Exception Types

- Sometimes none of the standard exception types describe your particular error condition well enough.

```
if (amount > balance) {
   throw new InsufficientFundsException(
      "withdrawal of " + amount +
      " exceeds balance of " + balance);
}
```

- It is a good idea to extend an appropriate class in the exception hierarchy.

Example Your Own Exception Types

```
public class InsufficientFundsException
    extends IllegalArgumentException
{
    public InsufficientFundsException() {}

    public InsufficientFundsException(String message)
    {
        super(message);
    }
}
```