Exceptions and Exception Handling

CS 240 – Advanced Programming Concepts

Exceptions Overview

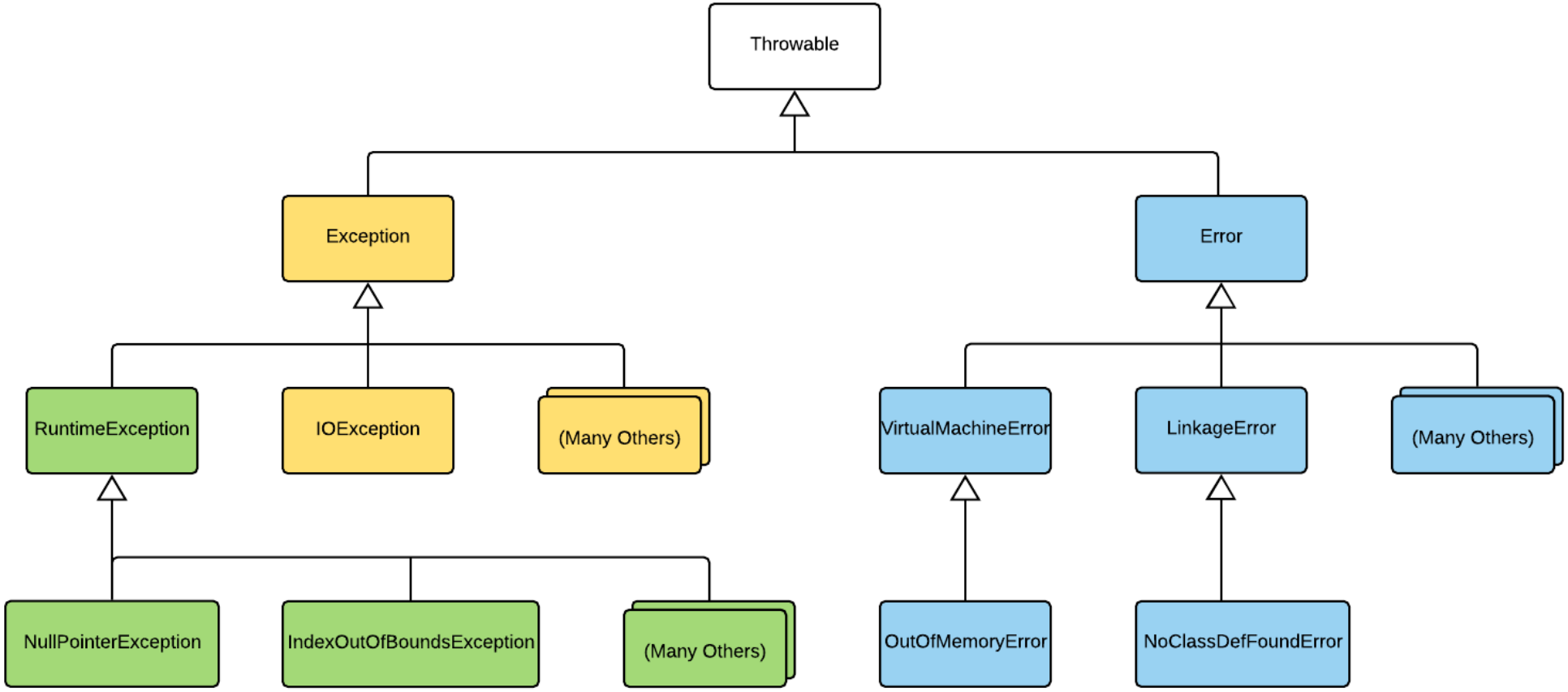
- Abnormal conditions that can occur in a Java class
- May be but are not necessarily errors
- Allow you to separate normal processing logic from abnormal processing logic
- Represented by classes and objects in Java

Separation of Normal and Abnormal Processing Logic

```
public class FileReadingWithoutExceptions {

   public boolean readFile(String fileName, String fileContents) {
      boolean success = true;

      File file = new File(filename);

      if(!file.exists()) {
         success = false;
      }

      if(success) {
         fileContents = "<Insert code to read the

            file here>";

         if(fileContents == "") {
            success = false;
         }
      }

      return success;
   }
}
```

```
public class FileReadingWithExceptions {

   public String readFile(String fileName) throws IOException {
      File file = new File(fileName);
      String fileContents = "<Insert code to read the

         file here>";
      return fileContents;
   }
}
```

Partial Exception Class Hierarchy



Try / Catch Blocks

Syntax

```
try {
  // Code that may throw an exception
} catch(SomeExceptionType ex) {
  // Code to handle the exception
} catch(OtherExceptionType ex) {
  // Code to handle the exception
}
```

Catching but not actually resolving an exception is sometimes called "swallowing" an exception and is normally considered a bad practice.

```
public class TryCatchExample {
    public String readFile(String fileName) {
        File file = new File(fileName);
        String fileContents = "";

        try {
            FileReader fr = new FileReader(file);
            BufferedReader br = new BufferedReader(fr);

            String line;
            while((line = br.readLine()) != null) {
                fileContents += line;
                fileContents += "\n";
            }

            return fileContents;

        } catch(IOException ex) {
            ex.printStackTrace();
            return fileContents;
        }
    }
}
```

Multi-Catch

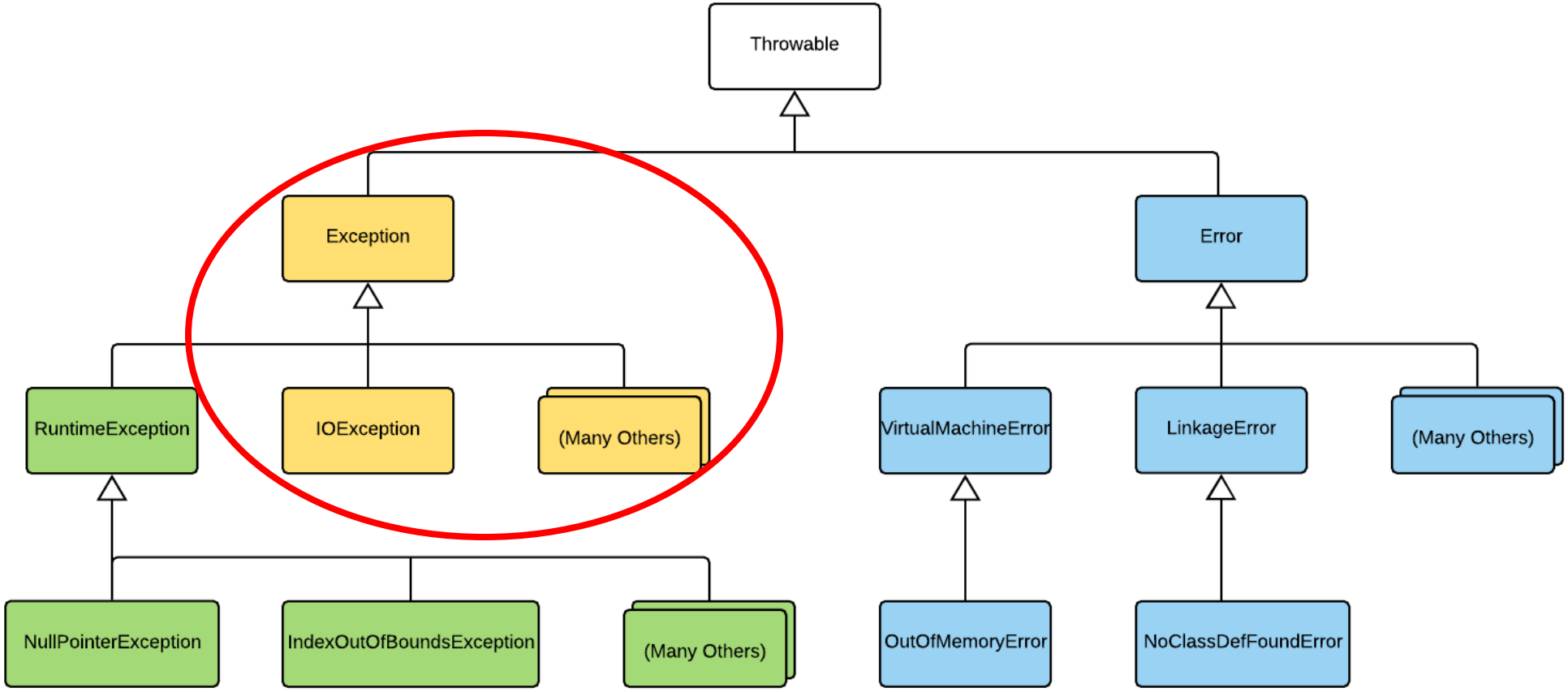
Without Multi-Catch

```
try {
   // Code that may throw an exception
} catch(SomeExceptionType ex) {
   // Code to handle the exception
} catch(OtherExceptionType ex) {
   // Code to handle the exception
}
```

With Multi-Catch (added in version 7)

```
try {
   // Code that may throw an exception
} catch(SomeExceptionType | OtherExeptionType ex) {
   // Code to handle the exception
}
```

Checked Exceptions



The Handle or Declare Rule

- Applies to Checked Exceptions (Exceptions that are not RuntimeExceptions)
- Does not apply to Errors
- Does not apply to RuntimeExceptions

Handle

```
try {
} catch(Exception ex) {
}
```

or

Declare

```
public void method throws Exception {
}
```

Finally Blocks

- Always executed
  - Whether an exception occurs or not
  - Whether a catch block exists or not
  - Whether a method is terminated because of an exception or not
- Normally used to clean up resources (i.e. close files, connections, etc.)

- Example:
  - FinallyExample.java

Try with Resources

```
public class TryWithResourcesExample {

    public String readFile(String fileName) throws IOException {
        File file = new File(fileName);
        String fileContents = "";

        try(FileReader fr = new FileReader(file);
            BufferedReader br = new BufferedReader(fr)) {

            String line;
            while((line = br.readLine()) != null) {
                fileContents += line;
                fileContents += "\n";
            }

            return fileContents;
        }
    }
}
```

Throwing an Exception

```
public class ExceptionThrowingExample {

    public void myMethod(int methodParam) {
        if(methodParam < 0) {
            throw new IllegalArgumentException("The parameter may not be negative");
        }
    }
}
```

Rethrowing an Exception

```
public class ExceptionRethrowingExample {

    public void myMethod() {
        try {
            // Code that may throw different types of exceptions
        } catch (RuntimeException ex) {
            throw ex;
        } catch (Exception ex) {
            // Do something with all checked exceptions
        }
    }
}
```

Avoid swallowing RuntimeExceptions

The Overridden Method Exception Rule

- Overriding methods **can not** throw checked exceptions that are not expected by callers of the overridden method
- Overriding methods **can** throw:
  - Fewer exceptions than the overridden method
  - The same exceptions as the overridden method
  - Subclasses of exceptions thrown by the overridden method
  - RuntimeExceptions and any RuntimeException subclasses
  - Errors and any Error subclasses
- Overriding methods **can not** throw:
  - Anything else (checked exceptions that are not subclasses of thrown exceptions)

Creating Your Own Exception Classes

```
public class ImageEditorException extends Exception {
    public ImageEditorException() {
    }

    public ImageEditorException(String message) {
        super(message);
    }

    public ImageEditorException(String message, Throwable cause) {
        super(message, cause);
    }

    public ImageEditorException(Throwable cause) {
        super(cause);
    }
}
```