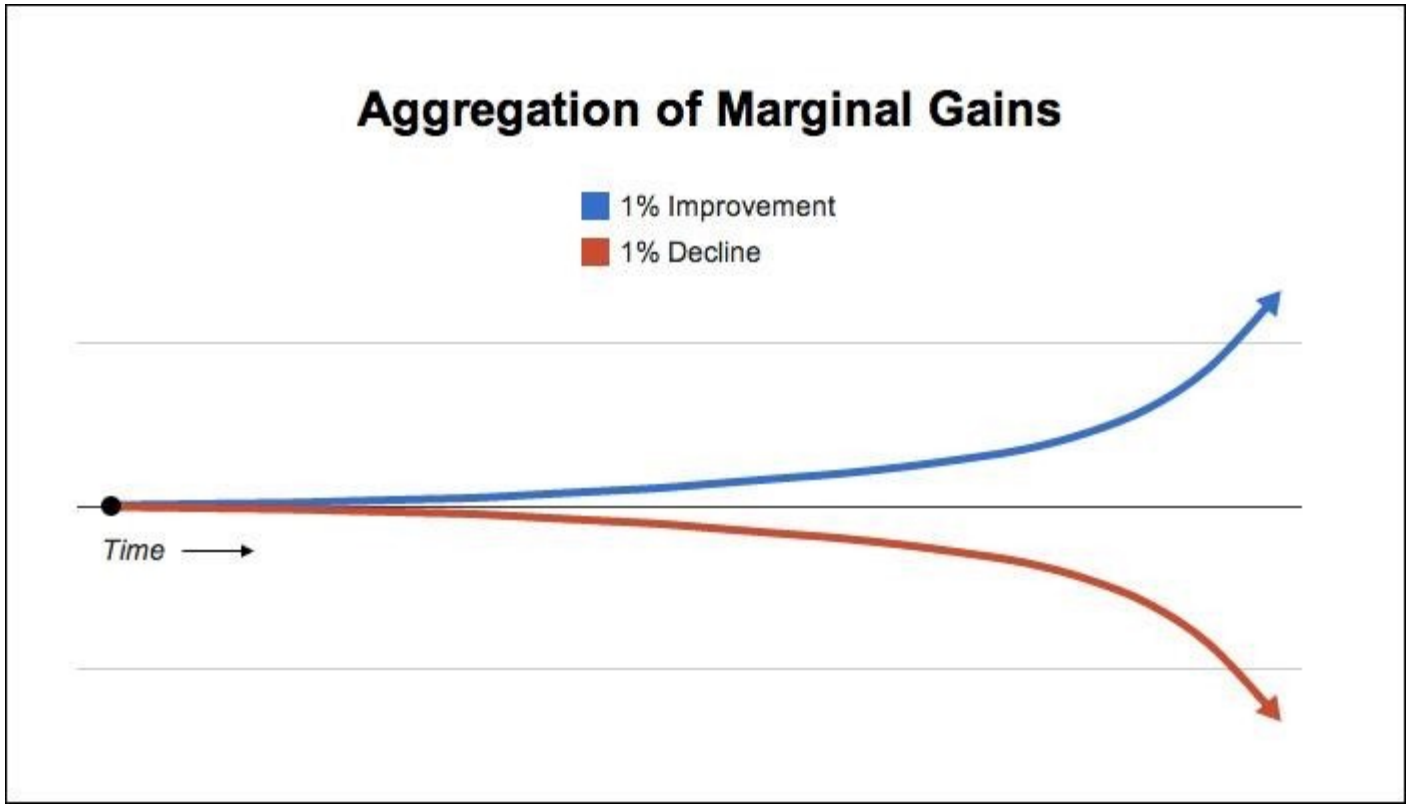
the UNIVERSITY of GREENWICH

COMP1618
Exception



the UNIVERSITY of GREENWICH

Lecture Objectives

Discuss the following main topics:

- Handling Exceptions
- Throwing Exceptions
- Input/Output Exceptions

```
try {

  // code that may throw an exception.
  :
} catch (ExceptionType parameterName) {

  // code to deal with the exception.
  :
}
```

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Program {
  public static void main(String[] args) {
    try {
      File file = new File("missing.txt");
      Scanner scanner = new Scanner(file);
      while (scanner.hasNextLine())
        System.out.println(scanner.nextLine());

      System.out.println("printing inside of try...");

    } catch (FileNotFoundException e) {
      System.err.println("File not found.");
    }

    System.out.println("printing outside of try...");
  }
}
```

the UNIVERSITY of GREENWICH

Handling Exceptions

- An exception is an object that is generated as the result of an error or an unexpected event.
- Exception are said to have been “thrown.”
- It is the programmers responsibility to write code that detects and handles exceptions.
- Unhandled exceptions will crash a program.
- Example: BadArray.java
- Java allows you to create exception handlers.

the UNIVERSITY of GREENWICH

Code that throws exceptions

dividing by zero:

```
int x = 0;
System.out.println(1 / x);  // ArithmeticException
```

trying to dereference a null variable:

```
Point p = null;
p.translate(2, -3);          // NullPointerException
```

trying to interpret input in the wrong way:

```
// NumberFormatException
int err = Integer.parseInt("hi");
```

reading a non-existent file:

```
// FileNotFoundException
Scanner in = new Scanner(new File("notHere.txt"));
```

the UNIVERSITY of GREENWICH

Exception avoidance

In many cases, the best plan is to try to avoid exceptions.

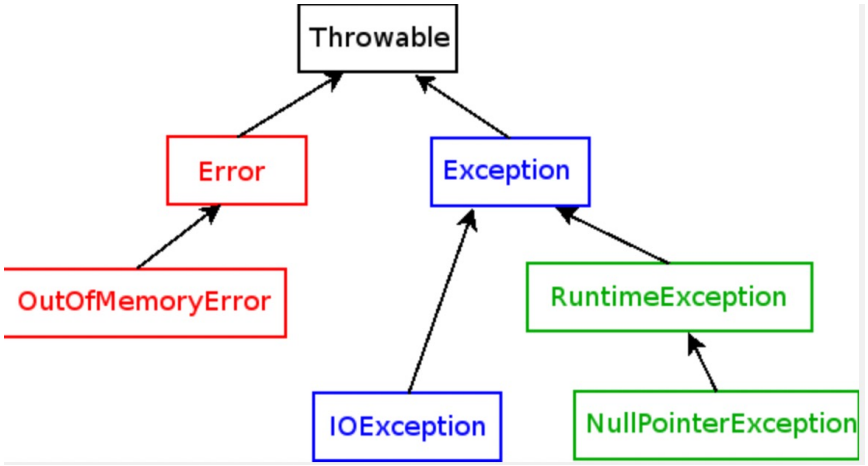
```
// better to check first than try/catch without check
int x;
...
if (x != 0) {
    System.out.println(1 / x);
}


File file = new File("notHere.txt");
if (file.exists()) {
    Scanner in = new Scanner(file);
}


// can we avoid this one?
int err = Integer.parseInt(str);
```

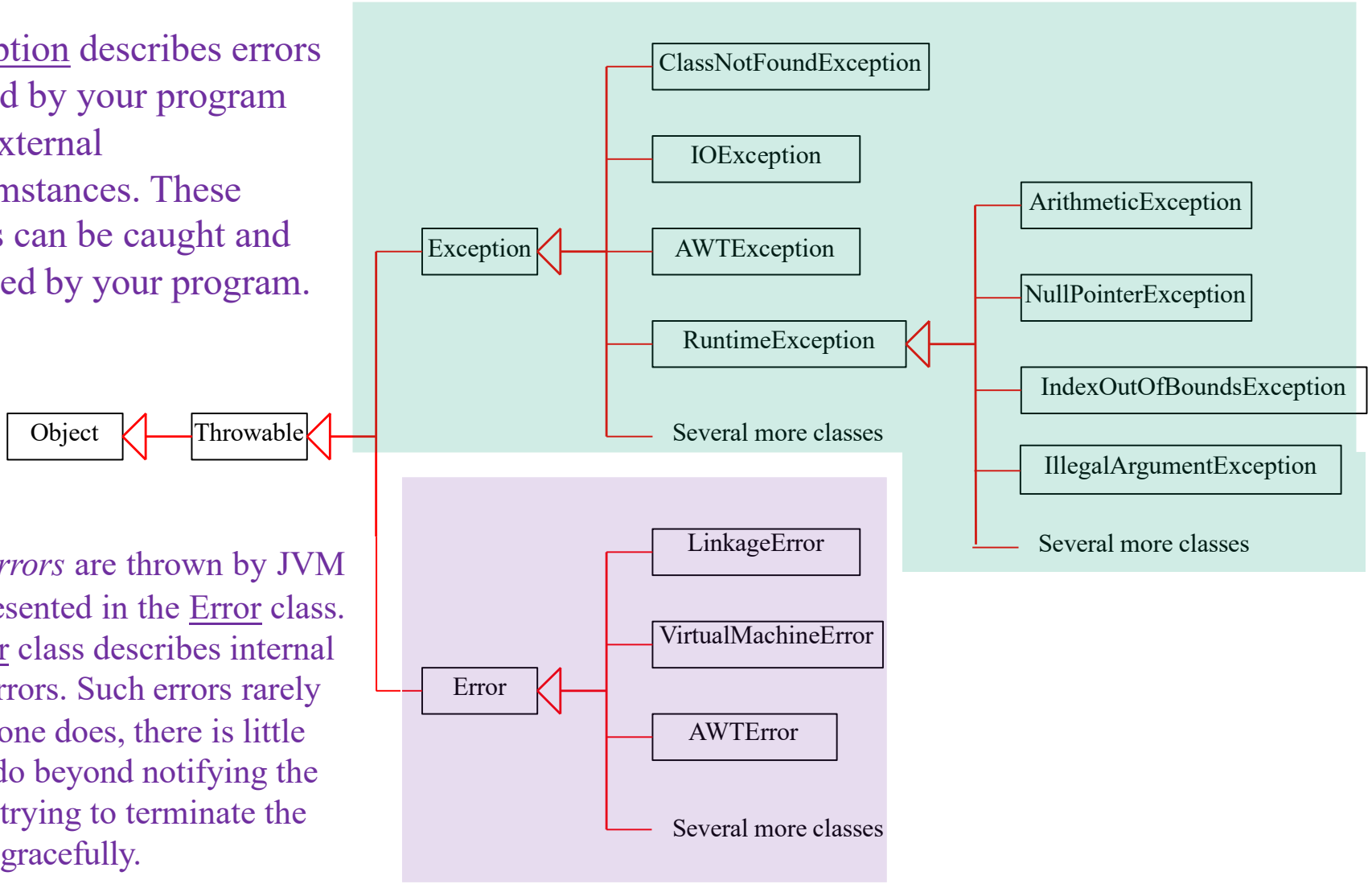
Exception Classes

- Exceptions are ordinary Java objects that are subclasses of the **Java.lang.Throwable** class.
- **Error** and subclasses of Error represent very serious error conditions, such as running out of memory.
- **Exception**, and all of its subclasses (except RuntimeException and its subclasses) represent errors that are generally **recoverable**.



Exceptions

Exception describes errors caused by your program and external circumstances. These errors can be caught and handled by your program.



System errors are thrown by JVM and represented in the Error class. The Error class describes internal system errors. Such errors rarely occur. If one does, there is little you can do beyond notifying the user and trying to terminate the program gracefully.

the UNIVERSITY of GREENWICH

How Exceptions Work

There are **three** ways that an exception can be thrown in some block of code:

- The block of code called a method which threw an exception
- One of the statements in the block of code did something bad, such as call a method on a variable storing a null value, or access an array element at an invalid index.
- The block of code explicitly threw an exception using a **throw** statement.

When an exception is thrown in some block of code in a method,

- Either the exception is **caught**,
- or the exception is thrown out of the method.

Handling Exceptions – try block

- To handle an exception, you use a *try* statement.

```
try
{
   (try block statements...)
}
catch (ExceptionType ParameterName)
{
   (catch block statements...)
}
```

- First the keyword `try` indicates a block of code will be attempted (the curly braces are required).
- This block of code is known as a *try block*.

Handling Exceptions – try block

- A *try block* is:
  - one or more statements that are executed, and
  - can potentially throw an exception.

- The application will **NOT halt** if the try block throws an exception.

- After the try block, a `catch` clause appears.

Handling Exceptions – *catch* block

- A catch clause begins with the key word `catch`:

  `catch (`*`ExceptionType ParameterName`*`)`

  - *`ExceptionType`* is the name of an exception class
  - *`ParameterName`* is a variable name which will reference the exception object if the code in the try block throws an exception.
- The code that immediately follows the catch clause is known as a *catch block* (the curly braces are required).
- The code in the catch block is executed if the try block throws an exception.

the UNIVERSITY of GREENWICH

Handling Exception - example

- This code is designed to handle a `FileNotFoundException` if it is thrown.

```
try
{
  File file = new File ("MyFile.txt");
  Scanner inputFile = new Scanner(file);
}
catch (FileNotFoundException e)
{
  System.out.println("File not found.");
}
```

- The Java Virtual Machine searches for a `catch` clause that can deal with the exception.
- Example: OpenFile.java

the UNIVERSITY of GREENWICH

Handling Exceptions

- The parameter must be of a type that is compatible with the thrown exception's type.

- After an exception, the program will continue execution at the point just past the catch block.

the UNIVERSITY of GREENWICH

Handling Exceptions

- Each exception object has a method named `getMessage` that can be used to retrieve the default error message for the exception.

- Example:
  - ExceptionMessage.java
  - ParseIntError.java

Exception methods

All exception objects have these methods:

Method	Description
`public String` **`getMessage`**`()`	text describing the error
`public String` **`toString`**`()`	exception's type and description
`getCause``(),` **`getStackTrace`**`(),` **`printStackTrace`**`()`	other methods

```
try {
    readFile();
} catch (IOException e) {
    System.out.println("I/O error: " +
e.getMessage());
}
```

the UNIVERSITY of GREENWICH

Don't ignore exceptions

Effective Java Tip: Don't ignore exceptions!

An empty `catch` block is (a common) poor style.

- often hide an error

```
try {
    readFile(filename);
} catch (IOException e) {}  // do nothing on error
```

At a *minimum*, print out the exception so you know it happened.

```
} catch (IOException e) {
    e.printStackTrace();    // just in case
}
```

the UNIVERSITY of GREENWICH

Multiple Exceptions

- The code in the *try block* may be capable of throwing more than one type of exception.
- A `catch` clause needs to be written for each type of exception that could potentially be thrown.
- The `catch` clauses must be listed from most specific to most general. Each `catch` block is tried in order.

- Example: SalesReport.java, SalesReport2.java

Catch multiple exceptions

```
try {
     statement(s);
} catch (type1 name) {
     code to handle the exception
} catch (type2 name) {
     code to handle the exception
  ...
} catch (typeN name) {
     code to handle the exception
}
```

You can catch more than one kind of exception in the same code. When an exception is thrown, the matching catch block (if any) is used.

If multiple `catch` blocks match, the most specific match is chosen.

the UNIVERSITY of GREENWICH

Exception Handlers

- The following two catch blocks try to catch the same type of exception (ERROR!)

```
try
{
  number = Integer.parseInt(str);
}
catch (NumberFormatException e)
{
  System.out.println("Bad number format.");
}
catch (NumberFormatException e) //  ERROR!!!
{
  System.out.println(str + " is not a number.");
}
```

the UNIVERSITY of GREENWICH

Exception Handlers

- The previous code could be rewritten to work, as follows, with no errors:

```
try
{
  number = Integer.parseInt(str);
}
catch (NumberFormatException e)
{
  System.out.println(str
                     + " is not a number.");
}
catch (IllegalArgumentException e) //OK
{
  System.out.println("Bad number format.");
}
```

The `finally` Clause

- The *try-catch* statement may have an optional `finally` clause.
- If so, the `finally` clause must appear after all of the `catch` clauses.

```
try
{
   (try block statements...)
}
catch (ExceptionType ParameterName)
{
   (catch block statements...)
}
finally
{
   (finally block statements...)
}
```

the UNIVERSITY of GREENWICH

The `finally` Clause

For each try block there can be zero or more catch blocks, but **only one finally block**.

Java *finally-block* is always executed no matter whether an exception is handled or not.

The finally block will **NOT** be executed if the program exits (either by calling *System.exit()* or by causing a fatal error that causes the process to abort).

the UNIVERSITY of GREENWICH

Example finally block -1

```
public static void main(String args[]){
    try {
     //below code throws Arithmetic type exception
     int data=25/0;
     System.out.println(data);
    }
    //can only accept Null Pointer type exception
    catch(NullPointerException e){
      System.out.println(e);
    }
    finally {
      System.out.println("finally block is always executed");

    }

    System.out.println("rest of the code...");
}
```

the UNIVERSITY of GREENWICH

Example finally block -2

```
public static void main(String args[]){
    try {
     //below code throws Arithmetic type exception
     int data=25/0;
     System.out.println(data);
    }
    //can only accept Null Pointer type exception
    catch(ArithmeticException e){
      System.out.println(e);
    }
    finally {
      System.out.println("finally block is always executed");

    }

    System.out.println("rest of the code...");
}
```

The *Throws* clause

```
public type name(parameters) throws type {
```

A clause in a method header claiming it may cause an exception.

Needed when a method may throw an uncaught checked exception.

```
public void processFile(String filename)
                    throws FileNotFoundException {
```

The above means one of two possibilities:

- `processFile` itself might throw an exception.
- `processFile` might call some sub-method that throws an exception, and it is choosing not to catch it (rather, to re-throw it out to the caller).

the UNIVERSITY of GREENWICH

Dealing with File IO Exceptions

- The `Scanner` class and `PrintWriter class` can throw an `IOException` when a `File` object is passed to its constructor.

- we can use **try /catch** blocks to handle the exception, or put a **`throws IOException`** clause in the header of the method that instantiates the `Scanner` class.

the UNIVERSITY of GREENWICH

Example: Throwing an exception

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

public class ReadFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      Scanner myReader = new Scanner(myObj);
      while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        System.out.println(data);
      }
      myReader.close();
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Example: Throwing an exception

```
public void writeFile(String fileName)
                                    throws IOException {
   ...
   PrintWriter out = new PrintWriter(fileName);
   out.print("Lab 3 results: ");
   out.println(result);
   out.close();

}
```

If there is a problem with opening the file for writing, `PrintWriter` will throw an `IOException`.

The method can throw this exception to the method that called it.

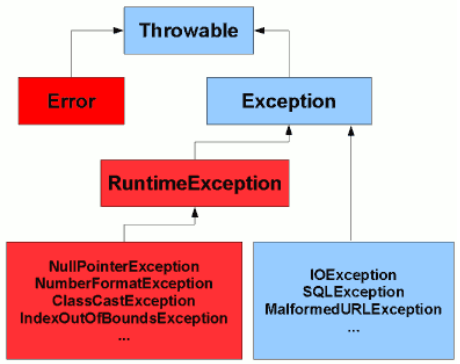
the UNIVERSITY of GREENWICH

Checked/Unchecked exceptions

Java has two major kinds of exceptions:

Checked exceptions: Ones that MUST be handled by a `try/catch` block (or `throws` clause) or else the program will not compile.

- Meant for serious problems that the caller ought to deal with.
- Subclasses of `Exception` in the inheritance tree.



Unchecked exceptions: Ones that don't have to be handled; if not handled, the program halts.

- Meant for smaller errors or programmer errors.
- Subclasses of `RuntimeException` in the tree.
- Mistakes that could have been avoided by a test.
  - check for null or 0, check if a file exists, check array's bounds, ...

the UNIVERSITY of GREENWICH

Summary

- We have looked at:
  - Exception: try, catch, finally
  - File I/O exceptions