Exception Handling

CSE 5910

www.eecs.yorku.ca/course/5910

Sources of Crashes

- The user

  Enter your choice (1–5): a

- The client

```
List<Integer> list = ...
for (int i = 0; i <= list.size(); i++) {
    output.println( list .get(i ));
}
```

- The implementer

```
import com.cheapbutquestionable.Integers;
 ...
int value = Integer.parseInt(input.nextInt());
```

- The runtime environment

```
List<String> list = ...
while (true) {
    list .add(new String("Hello"));
```

API

Which exceptions a method may throw are specified in the API.

`E get(int index)`

Returns the element at the specified position in this list.

Parameters:

`index` – index of the element to return

Returns:

the element at the specified position in this list

Throws:

`IndexOutOfBoundsException` – if the index is out of range (`index < 0 || index >= size()`)

Preconditions versus Exceptions

Question

Why do we need exceptions? Can't we prevent crashes by introducing appropriate preconditions?

Preconditions versus Exceptions

Question

Why do we need exceptions? Can't we prevent crashes by introducing appropriate preconditions?

Answer

Introducing an appropriate precondition is not always practical and in some cases impossible.

Preconditions versus Exceptions

The method Double.valueOf(String) throws a NumberFormatException if the argument is not a parsable number.

If this exception were replaced with a precondition, the client would have to check that the argument is a parsable number. Although this can be done using a regular expression, as shown in the API of the method Double.valueOf(String), using exception handling is much easier.

Preconditions versus Exceptions

Each constructor throws an OutOfMemoryError when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.

If this error were replaced with a precondition, the client would have to check if there would be sufficient memory before creating each object, which is obviously extremely tedious (if at all possible).

How to Handle Exceptions

Step 1

Place a try block around the statement(s) that may throw the exception.

```
try {
    ...
}
```

Step 2

Place a catch block right after the try block.

```
catch  (... Exception e) {
    ...
}
```

Exceptions

Compiling

```
File file = new File("test.txt");
PrintStream fileOutput = new PrintStream(file);
```

gives rise to the error

```
Client.java:13: unreported exception java.io.
FileNotFoundException; must be caught or declared
to be thrown
    PrintStream fileOutput = new PrintStream(file);
                             ^
1 error
```

Why?

Answer

Because the constructor `PrintStream(File)` throws a `FileNotFoundException` if the `file` object does not denote an existing, writable regular file and a new regular file of that name cannot be created, or if some other error occurs while opening or creating the file (see API).

Answer

Because the constructor `PrintStream(File)` throws a `FileNotFoundException` if the `file` object does not denote an existing, writable regular file and a new regular file of that name cannot be created, or if some other error occurs while opening or creating the file (see API).

Question

How does a client fix a "must be caught or declared to be thrown" error?

Answer

Because the constructor `PrintStream(File)` throws a `FileNotFoundException` if the `file` object does not denote an existing, writable regular file and a new regular file of that name cannot be created, or if some other error occurs while opening or creating the file (see API).

Question

How does a client fix a "must be caught or declared to be thrown" error?

Answer

Catch the exception. (An implementer may also decide the declare the exception to be thrown.)

How to Catch Exceptions?

```
import java.io.FileNotFoundException;
 ...
try{
    File file = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
```

If all goes well

```
import java.io.FileNotFoundException;
 ...
try{
    File  file  = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If all goes well

```
import java.io.FileNotFoundException;
 ...
try{
    File file = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If all goes well

```
import java.io.FileNotFoundException;
 ...
try{
    File file = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If all goes well

```
import java.io.FileNotFoundException;
 ...
try{
    File  file  = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If all goes well

```
import java.io.FileNotFoundException;
 ...
try{
    File  file  = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If the file does not exist

```
import java.io.FileNotFoundException;
 ...
try{
    File  file  = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If the file does not exist

```
import java.io.FileNotFoundException;
 ...
try{
    File file = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If the file does not exist

```
import java.io.FileNotFoundException;
 ...
try{
    File file = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If the file does not exist

```
import java.io.FileNotFoundException;
 ...
try{
    File  file  = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If the file does not exist

```
import java.io.FileNotFoundException;
 ...
try{
    File   file   = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file: "
        + e.getMessage())
}
...
```

If the file does not exist

```
import java.io.FileNotFoundException;
 ...
try{
    File  file  = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If the file name is null

```
import java.io.FileNotFoundException;
 ...
try{
    File  file  = new File(null);
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If the file name is null

```
import java.io.FileNotFoundException;
 ...
try{
    File file = new File(null);
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If the file name is null

```
import java.io.FileNotFoundException;
 ...
try{
    File  file  = new File("null");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

If the file name is null

```
import java.io.FileNotFoundException;
 ...
try{
    File  file  = new File("test.txt");
    PrintStream fileOutput = new PrintStream(file);
    ...
}
catch (FileNotFoundException e){
    output.println("Failed to write to file : "
        + e.getMessage())
}
...
```

Since a NullPointerException is not a FileNotFoundException, the app crashes.

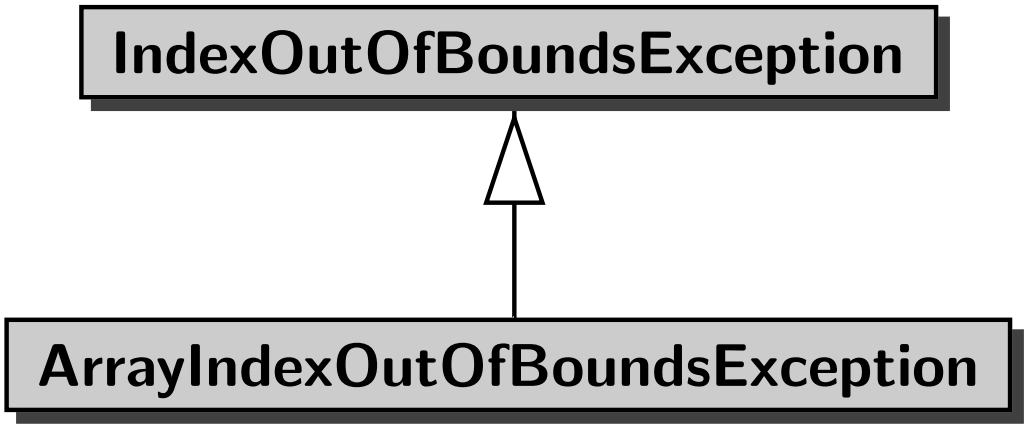
Exceptions

```
try{
    output.println(args [0]);
}
catch (IndexOutOfBoundsException e){
    output.println(e.getMessage());
}
catch (ArrayIndexOutOfBoundsException e){
    e.printStackTrace();
}
```

gives rise to the compile-time error

```
Client.java:19: exception java.lang.
ArrayIndexOutOfBoundsException has already been
caught
    catch (ArrayIndexOutOfBoundsException e)
    ^
1 error
```

Exceptions

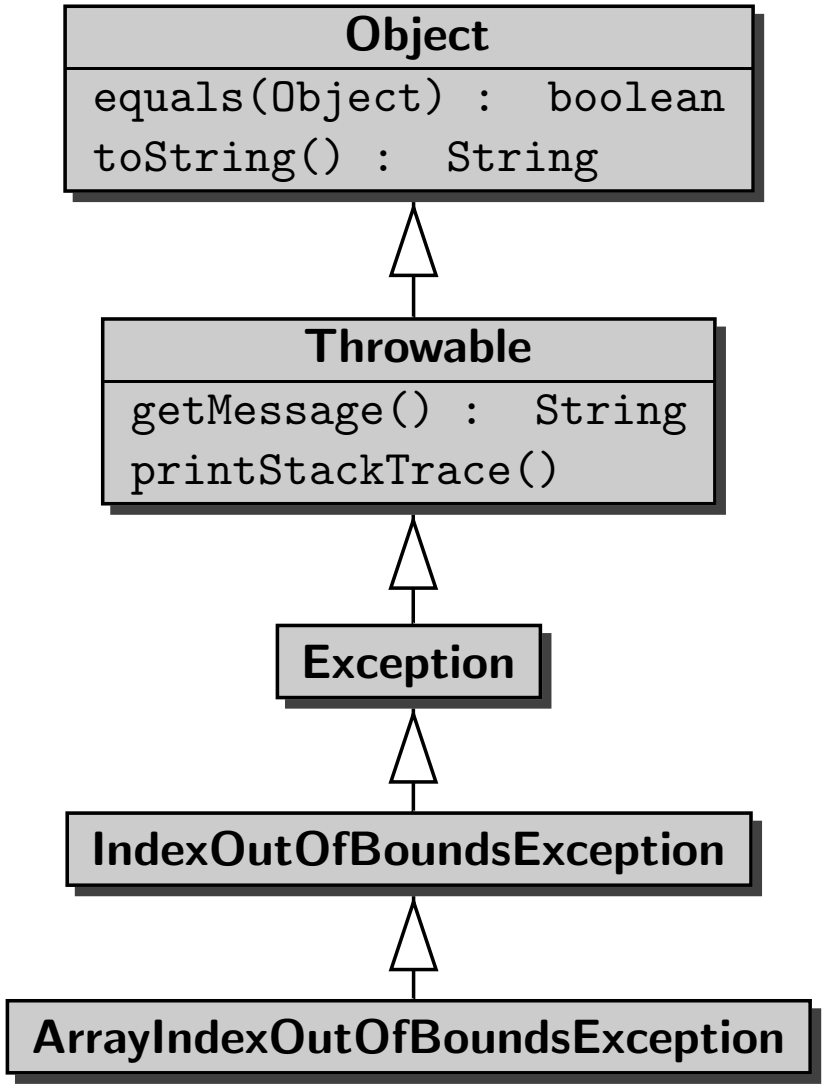

IndexOutOfBoundsException
ArrayIndexOutOfBoundsException

Exceptions

```
try{
    output.println(args [0]);
}
catch (IndexOutOfBoundsException e){
    output.println(e.getMessage());
}
catch (ArrayIndexOutOfBoundsException e){
    e.printStackTrace();
}
```

The second catch block is redundant, because an `ArrayIndexOutOfBoundsException` is-an `IndexOutOfBoundsException`.

Inheritance Hierarchy



Question

May the method `charAt(int)` of the class `String` throw an exception?

Question

May the method `charAt(int)` of the class `String` throw an exception?

Answer

Yes.

Exceptions

Question

May the method `charAt(int)` of the class `String` throw an exception?

Answer

Yes.

Question

Which type of exception?

Exceptions

Question

May the method `charAt(int)` of the class `String` throw an exception?

Answer

Yes.

Question

Which type of exception?

Answer

An `IndexOutOfBoundsException`.

Exceptions

```
String word = ...;
output.println(word.charAt(2));
```

Question

Why does the above snippet not give rise to a "must be caught or declared to be thrown" error?

```
String word = ...;
output.println(word.charAt(2));
```

Question

Why does the above snippet not give rise to a "must be caught or declared to be thrown" error?

Answer

The "must be caught or declared to be thrown" rule is only applicable to checked exceptions and an `IndexOutOfBoundsException` is not checked.

What are Checked Exceptions?

Definition

An exception is checked if

- it is `Exception` or any of its subclasses, and
- it is not `RuntimeException` or any of its subclasses.

Question

Is `NullPointerException` checked?

What are Checked Exceptions?

Definition

An exception is checked if

- it is `Exception` or any of its subclasses, and
- it is not `RuntimeException` or any of its subclasses.

Question

Is `NullPointerException` checked?

Answer

No.

What are Checked Exceptions?

Definition

An exception is checked if

- it is `Exception` or any of its subclasses, and
- it is not `RuntimeException` or any of its subclasses.

Question

Is `NullPointerException` checked?

Answer

No.

Question

Is `InvalidPropertiesFormatException` checked?

What are Checked Exceptions?

Definition

An exception is checked if

- it is `Exception` or any of its subclasses, and
- it is not `RuntimeException` or any of its subclasses.

Question

Is `NullPointerException` checked?

Answer

No.

Question

Is `InvalidPropertiesFormatException` checked?

Answer

Yes.

What are Checked Exceptions?

Definition

An exception is checked if

- it is `Exception` or any of its subclasses, and
- it is not `RuntimeException` or any of its subclasses.

Question

Is `Exception` checked?

What are Checked Exceptions?

Definition

An exception is checked if

- it is `Exception` or any of its subclasses, and
- it is not `RuntimeException` or any of its subclasses.

Question

Is `Exception` checked?

Answer

Yes.

What are Checked Exceptions?

Definition

An exception is checked if

- it is `Exception` or any of its subclasses, and
- it is not `RuntimeException` or any of its subclasses.

Question

Is `Exception` checked?

Answer

Yes.

Question

Is `RuntimeException` checked?

What are Checked Exceptions?

Definition

An exception is checked if

- it is `Exception` or any of its subclasses, and
- it is not `RuntimeException` or any of its subclasses.

Question

Is `Exception` checked?

Answer

Yes.

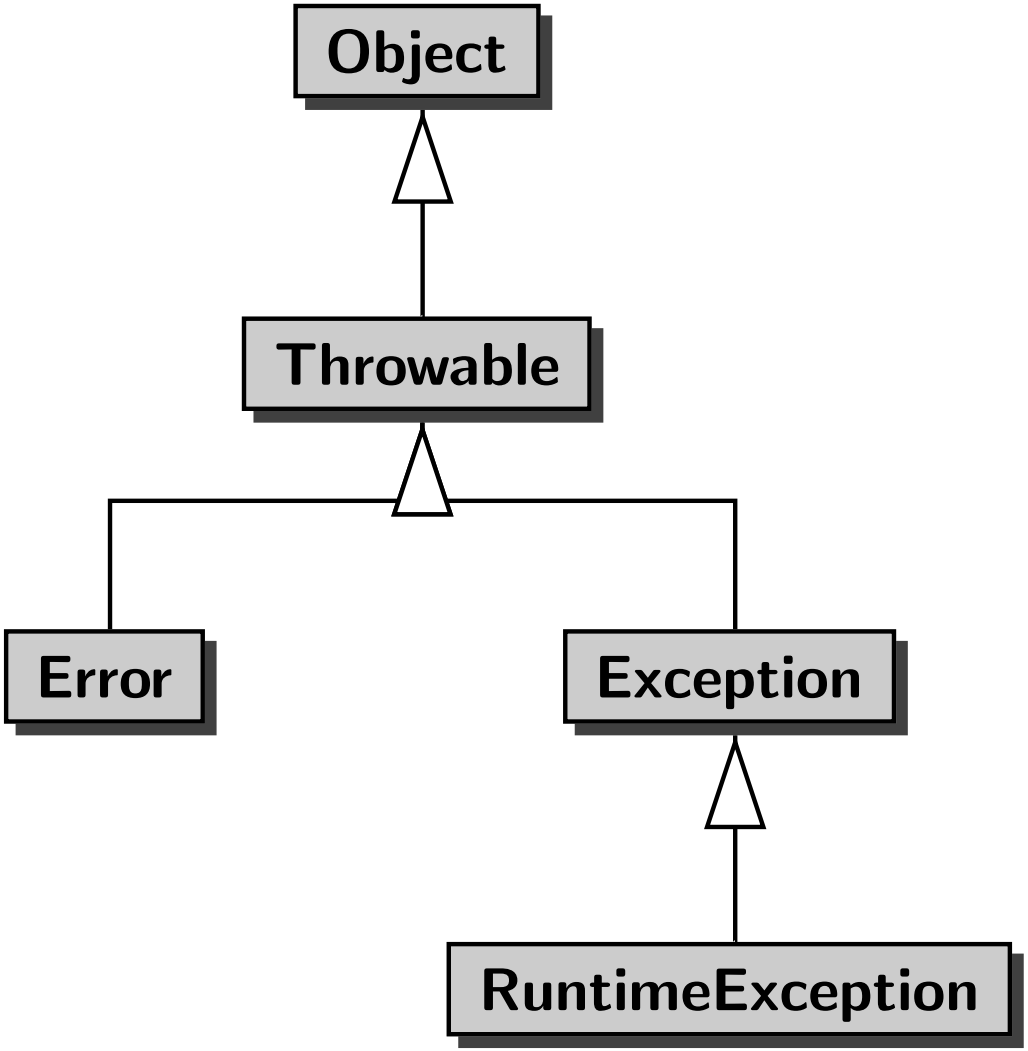
Question

Is `RuntimeException` checked?

Answer

No.

Inheritance Hierarchy



Errors

Question

Why are `Errors` exempt from the "must be caught or declared to be thrown" rule?

Errors

Question

Why are `Errors` exempt from the "must be caught or declared to be thrown" rule?

Answer

`Errors` represent conditions that are so abnormal the reliability of the whole environment is suspect and, hence, the code in the catch block may not run properly either.

Errors

Question

Why are `Errors` exempt from the "must be caught or declared to be thrown" rule?

Answer

`Errors` represent conditions that are so abnormal the reliability of the whole environment is suspect and, hence, the code in the catch block may not run properly either.

Question

Why are `RuntimeExceptions` exempt from the "must be caught or declared to be thrown" rule?

Errors

Question

Why are `Errors` exempt from the "must be caught or declared to be thrown" rule?

Answer

`Errors` represent conditions that are so abnormal the reliability of the whole environment is suspect and, hence, the code in the catch block may not run properly either.

Question

Why are `RuntimeExceptions` exempt from the "must be caught or declared to be thrown" rule?

Answer

`RuntimeExceptions` represent conditions that can be validated by the client.

Throwing Exceptions

Question

How can the client throw an exception?

Throwing Exceptions

Question

How can the client throw an exception?

Answer

```
throw new ...Exception(...);
```

Throwing Exceptions

Question

How can the client throw an exception?

Answer

```
throw new ...Exception(...);
```

Question

Why would a client ever throw an exception?

Throwing Exceptions

Question

How can the client throw an exception?

Answer

```
throw new ...Exception(...);
```

Question

Why would a client ever throw an exception?

Answer

For example, the client may want to separate the error handling code from the rest.