Programming for Mobile Computing

EECS 1022

`moodle.yorku.ca`

Drop deadline

July 22–31

During this period you can still drop the course but you will receive a W on your transcript. The W will not affect your gpa.

www.registrar.yorku.ca/enrol/dates/su17.htm contains important dates.

Final exam (written part)

When: Friday August 4, 14:00-15:30

Where:

- Curtis Lecture Hall E if your last name starts with A-K
- Curtis Lecture Hall F if your last name starts with L-Z

Material: everything covered in the course

Final exam (written part)

- No questions are allowed during the test. If a question is not clear, then write down any assumptions made.
- One page of notes (letter size, double sided) may be used during the test.
- A non-electronic dictionary may be used during the test.

Final exam (written part)

Preparation

- Study the material.
- Prepare your page of notes.
- Think of a test question.
- Post your question on the forum at Moodle.
- Answer questions posted by other students on the forum.
- Discuss questions and answers on the forum.

Extra labs

- Tuesday August 1, 17:30-19:30
- Thursday August 3, 17:30-19:30

Sources of Crashes

- Enter your choice (1–5): a

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

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

- 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()`)

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?

Exceptions

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 do we 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 do we fix a "must be caught or declared to be thrown" error?

Answer

We can catch the exception.

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())
}
...
```

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

Exceptions

Question

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

Exceptions

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?

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?

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.

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 programmer.

Throwing Exceptions

Question

How can we throw an exception?

Throwing Exceptions

Question

How can we throw an exception?

Answer

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

Throwing Exceptions

Question

How can we throw an exception?

Answer

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

Question

Why would a programmer ever throw an exception?

Throwing Exceptions

Question

How can we throw an exception?

Answer

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

Question

Why would a programmer ever throw an exception?

Answer

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