COMP122 Week 9

Exception Handling

UNIVERSITY OF LIVERPOOL

Dr. Patrick Totzke
totzke@liverpool.ac.uk

https://liverpool.instructure.com/courses/59716

Exception Handling

An Exception is an anomalous situation that requires special processing.

The concept pre-dates OOP

- in hardware: traps, interrupts (COMP109)
- in software: fairly old concept, already discussed for LISP in the 60s.

Many Languages have built-in support

- to standardise error handling (compiler support,...)
- clean separation of error handling and normal code

Exceptions Handling – General Pattern in software

- you detect an anomaly and report it to your runtime environment: and exception is raised (python lingo) or thrown (Java).
- the environment looks for an exception handler that can catch and deal with it
- depending on what the handler does, the normal execution is resumed in a well-defined state.

Common code pattern in pseudocode

```
1  try {
2      do stuff;       // dangerous code.. errors may happen
3  }
4  catch exception {
5      deal with it;   // handler
6  }
```

What should the handler do? report and re-throw? log and continue?

Exception Handling in OOP Languages

In OOP languages including Java,

Exceptions are Objects

When an anomalous situation occurs

1. an `Exception` object is created. It represents the situation and contains information about what happened.
2. the interpreter looks for, and calls a matching handler.
3. If none is found the program will terminate.

Exceptions and Errors in Java

Java distinguishes Errors from Exceptions (both extend `Throwable`):

Errors

are serious problems, typically caused by the environment. They cannot (or should not) be dealt with by an application. Example: running out of memory.

Exceptions

are generally caused by the application itself and can (often) be recovered from. Example: attempt to divide by zero.

Checked vs. Unchecked Exceptions

In Java, Exceptions are either *checked* or *unchecked*.

Checked Exceptions

are ones for which the compiler ensures that we do not overlook them:

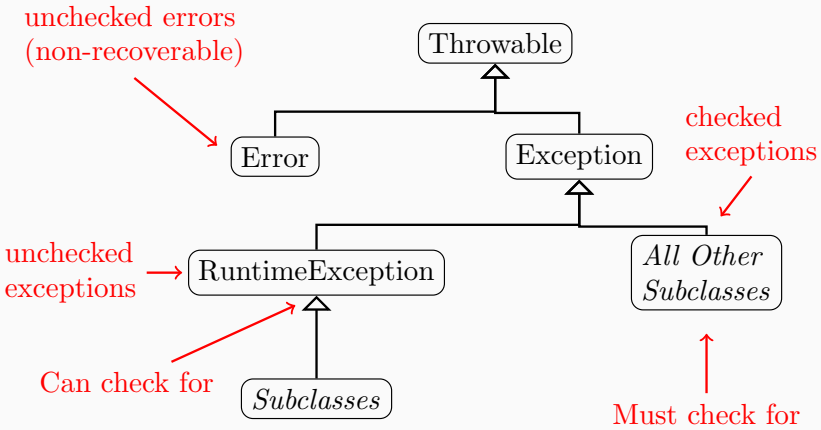
When calling a method that may throw a checked Exception, you are *required* to either catch and handle it, or declare that your code may throw an Exception of this type.

In other words, it is a compilation error (so the Java source file won't be compiled) if a checked exception is not handled in the application.

Unchecked Exception

May be thrown or handled without the compiler enforcing it. An exception is unchecked iff it extends `RuntimeException`.

Java – Throwable, Error and Exception classes



This, and the following examples are from Parsons, *Foundational Java*.

Checked Exceptions – A First Example

Here is a first example illustrating a possible exception caused by keyboard input

```
1  public class ReadFromKeyboard{
2
3     public static void main (String args[]) {
4
5        // System.in.read() reads one byte from stdin
6        int myChar = System.in.read();
7        System.out.println((char) myChar);
8     }
9  }
```

Checked Exceptions – A First Example

Here is a first example illustrating a possible exception caused by keyboard input

```
1  public class ReadFromKeyboard{
2
3     public static void main (String args[]) {
4
5        // System.in.read() reads one byte from stdin
6        int myChar = System.in.read();
7        System.out.println((char) myChar);
8     }
9  }
```

```
$ javac ReadFromKeyboard.java
ReadFromKeyboard.java:18: error: unreported exception IOException;
   must be caught or declared to be thrown
      int myChar = System.in.read();
                                  ^
1 error
```

Comments

- The class `ReadFromKeyboard` does not compile because `System.in.read()` can throw an `IOException`.
- The problem is that `IOException` is a checked Exception and program doesn't catch it nor declares that such Exceptions could arise (and passed upwards to a caller).

You have two options here:

1. To introduce an exception handler
2. Not to handle it yourself and declare that the `main` method may throw an `IOException`.

Re-throwing Exceptions

```
1  // introduce "IOException" to the namespace
2  import java.io.IOException;
3
4  public class ReadFromKeyboard2 {
5
6     public static void main (String args[]) throws IOException {
7
8        // The next line can throw an IOException
9        int myChar = System.in.read();
10
11       // Interpret that int as a char.  This will only
12       // be executed if the line above was successful.
13       System.out.println((char) myChar);
14    }
15 }
```

Now this program compiles with no problem.

Catching an Exception

An alternative would be to ringfence the offending line of code in a `try`/`catch` block to deal with this possible exception:

```
1  import java.io.IOException;
2
3  public class ReadFromKeyboard3 {
4
5     public static void main (String args[]) {
6
7        try{
8              int myChar = System.in.read();  // dangerous code
9              System.out.println((char) myChar);
10       }
11       catch(IOException e)  {
12             e.printStackTrace();
13       }
14    }
15 }
```

Comments

- Java's keywords for introducing potentially dangerous code and exception handlers are `try` and `catch` respectively.
- A `catch` block must follow a `try` block.
- There can be multiple exception handlers for one `try` block.
- each `catch` declares which type of exception it handles. This is used by the interpreter when searching for an appropriate handler!
- the `catch` block is handed an `Exception` object (in this case called `e`), that represents the error.

Try/Catch Blocks

The general syntax for try/catch blocks is as follows.

```
try{
  // dangerous code that may throw XException, YException, ...
}
catch(XException e)  {
  // handle XExceptions
}
catch(YException e)  {
  // handle YExceptions
}
...
finally{
  // will eventually get executed in any case
}
```

Unchecked Exceptions

In the previous program, `System.in.read()` is a method that reads from "standard input", one byte at a time, and can thrown (checked) `IOException`.

Alternatively, we've been using `java.io.Scanner` to collect input, e.g. using methods

- `int nextInt()`
- `long nextLong()`
- `double nextDouble()` ...

These do not throw checked exceptions but they may throw `RuntimeException`s when they try to parse the "wrong" kind of input.

Unchecked Exception – Example

```java
import java.util.Scanner;

public class ReadLong {

   public static void main (String args[]) {

      //  Create a Scanner instance
      Scanner scan = new Scanner(System.in);

      System.out.print("Enter an integer (or long): ");

      //  Try to get a long as input.  This can create an exception
      //  if something other than numbers are entered, or if
      //  a number with a decimal point is input, etc.
      long l = scan.nextLong();
      System.out.println(l);
   }
}
```

Trying out the program...

This compiles and behaves as expected on some “nice” kinds of input

```
$ java ReadLong
Enter an integer (or long): 43234
43234
```

...but not on others...

```
$ java ReadLong
Enter an integer (or long): fdafdas
Exception in thread "main" java.util.InputMismatchException
   at java.util.Scanner.throwFor(Scanner.java:864)
   at java.util.Scanner.next(Scanner.java:1485)
   at java.util.Scanner.nextLong(Scanner.java:2222)
   at java.util.Scanner.nextLong(Scanner.java:2182)
   at ReadLong.main(ReadLong.java:19)
$
```

Testing for an “Input Mismatch”

```
1  import java.util.Scanner;
2  import java.util.InputMismatchException;
3
4  public class ReadLong2 {
5
6     public static void main (String args[]) {
7
8        Scanner scan = new Scanner(System.in);
9        System.out.print("Enter an integer (or long): ");
10
11       try {
12              long l = scan.nextLong();
13              System.out.println(l);
14       }
15       catch (InputMismatchException e)  {
16              System.out.println("Oops! That input wasn't valid!");
17       }
18    }
19 }
```

Testing for an “Input Mismatch”

```
$ java ReadLong2
Enter an integer (or long): 4454
4454
```

```
$ java ReadLong2
Enter an integer (or long): fdfd
    Oops!  That input wasn't valid!
$
```

Self Study

As a (non-assessed) exercise, modify the last program to repeatedly ask for input from a user until you get valid input (a `long`)? What if you want a non-negative value?

A longer Example

Classifying user input (I)

```
1  /** ClassifyInput.java will read input from a user and
2   * attempt to parse that input as an int, long, float,
3   * double, and boolean.  This will output corresponding
4   * results to the user.
5   *
6   * @author Russell Martin, March 2018
7   */
8  import java.util.*;
9
10 public class ClassifyInput {
11
12    public static void main (String args[]) {
13
14       String testString;
15       Scanner input = new Scanner(System.in);
16       System.out.print("Type some input and I will attempt to classify
             what it is: ");
17
18       //  nextLine() always reads input as a java String.
```

Classifying user input (II)

```
19          testString = input.nextLine();
20
21          //  So we can try to convert the String to different formats...
22          try {
23             int i = Integer.parseInt(testString);
24             System.out.println("Input can be parsed as an int (in base 10):
                    "  + i);
25          }
26          catch (NumberFormatException e)  {
27             System.out.println("Input cannot be parsed as an int (in base
                   10).");
28          }
29
30          try {
31             int i = Integer.parseInt(testString, 16);
32             System.out.println("Input can be parsed as an int (translated
                   from base 16):  " + i);
33          }
34          catch (NumberFormatException e)  {
```

Classifying user input (III)

```
35            System.out.println("Input cannot be parsed as an int (translated
                  from base 16).");
36        }
37
38        try {
39            long l = Long.parseLong(testString);
40            System.out.println("Input can be parsed as a long (in base 10):
                  " + l);
41        }
42        catch (NumberFormatException e)  {
43            System.out.println("Input cannot be parsed as a long (in base
                  10).");
44        }
45
46        try {
47            long l = Long.parseLong(testString, 2);
48            System.out.println("Input can be parsed as a long (translated
                  from base 2):  " + l);
49        }
50        catch (NumberFormatException e)  {
```

Classifying user input (IV)

```
51            System.out.println("Input cannot be parsed as a long (translated
                  from base 2).");
52        }
53
54        try {
55            double d = Double.parseDouble(testString);
56            System.out.println("Input can be parsed as a double:  " + d);
57        }
58        catch (NumberFormatException e)  {
59            System.out.println("Input cannot be parsed as a double.");
60        }
61
62        try {
63            float f = Float.parseFloat(testString);
64            System.out.println("Input can be parsed as a float:  " + f);
65        }
66        catch (NumberFormatException e)  {
67            System.out.println("Input cannot be parsed as a float.");
68        }
69
```

Classifying user input (V)

```
70          //  parseBoolean always gives a true/false value.  It's only "true"
71          //  if the String is equal to "true" (ignoring case), otherwise it
72          //  returns "false".
73          boolean b = Boolean.parseBoolean(testString);
74          System.out.println("Input is parsed as a boolean with value:  " + b
                );
75
76      }  /*  End of "main" method  */
77
78  }
```

Sample classification output

```
$ java ClassifyInput
Type some input and I will attempt to classify what it is: 1011010
Input can be parsed as an int (in base 10):  1011010
Input can be parsed as an int (translated from base 16):  16846864
Input can be parsed as a long (in base 10):  1011010
Input can be parsed as a long (translated from base 2):  90
Input can be parsed as a double:  1011010.0
Input can be parsed as a float:  1011010.0
Input is parsed as a boolean with value:  false
```

```
$ java ClassifyInput
Type some input and I will attempt to classify what it is: 21.55e40
Input cannot be parsed as an int (in base 10).
Input cannot be parsed as an int (translated from base 16).
Input cannot be parsed as a long (in base 10).
Input cannot be parsed as a long (translated from base 2).
Input can be parsed as a double:  2.155E41
Input can be parsed as a float:  Infinity
Input is parsed as a boolean with value:  false
```

Sample classification output (cont.)

```
$ java ClassifyInput
Type some input and I will attempt to classify what it is: true
Input cannot be parsed as an int (in base 10).
Input cannot be parsed as an int (translated from base 16).
Input cannot be parsed as a long (in base 10).
Input cannot be parsed as a long (translated from base 2).
Input cannot be parsed as a double.
Input cannot be parsed as a float.
Input is parsed as a boolean with value:  true
```

```
$ java ClassifyInput
Type some input and I will attempt to classify what it is: 3424.555.65
Input cannot be parsed as an int (in base 10).
Input cannot be parsed as an int (translated from base 16).
Input cannot be parsed as a long (in base 10).
Input cannot be parsed as a long (translated from base 2).
Input cannot be parsed as a double.
Input cannot be parsed as a float.
Input is parsed as a boolean with value:  false
```

Quiz!

Quiz/Recap – Java Keywords Related to Exception Handling

`try`

Quiz/Recap – Java Keywords Related to Exception Handling

`try` ... Encloses a block of code that could throw an exception.

Quiz/Recap – Java Keywords Related to Exception Handling

`try` ... Encloses a block of code that could throw an exception.

`catch`

Quiz/Recap – Java Keywords Related to Exception Handling

`try` ... Encloses a block of code that could throw an exception.

`catch` ... follows a `try` block, encloses code that can be executed to respond to particular exception.

Quiz/Recap – Java Keywords Related to Exception Handling

`try` ... Encloses a block of code that could throw an exception.

`catch` ... follows a `try` block, encloses code that can be executed to respond to particular exception.

`finally`

Quiz/Recap – Java Keywords Related to Exception Handling

`try` ... Encloses a block of code that could throw an exception.

`catch` ... follows a `try` block, encloses code that can be executed to respond to particular exception.

`finally` ... follows a `try` block and (typically) one or more `catch` blocks, encloses a block of code that will execute regardless of whether any exceptions are thrown.

Quiz/Recap – Java Keywords Related to Exception Handling

`try` ... Encloses a block of code that could throw an exception.

`catch` ... follows a `try` block, encloses code that can be executed to respond to particular exception.

`finally` ... follows a `try` block and (typically) one or more `catch` blocks, encloses a block of code that will execute regardless of whether any exceptions are thrown.

`throws`

Quiz/Recap – Java Keywords Related to Exception Handling

`try` ... Encloses a block of code that could throw an exception.

`catch` ... follows a `try` block, encloses code that can be executed to respond to particular exception.

`finally` ... follows a `try` block and (typically) one or more `catch` blocks, encloses a block of code that will execute regardless of whether any exceptions are thrown.

`throws` ... Indicates that a method may throw an exception.

Quiz/Recap – Java Keywords Related to Exception Handling

`try` ... Encloses a block of code that could throw an exception.

`catch` ... follows a `try` block, encloses code that can be executed to respond to particular exception.

`finally` ... follows a `try` block and (typically) one or more `catch` blocks, encloses a block of code that will execute regardless of whether any exceptions are thrown.

`throws` ... Indicates that a method may throw an exception.

`throw`

Quiz/Recap – Java Keywords Related to Exception Handling

`try` ... Encloses a block of code that could throw an exception.

`catch` ... follows a `try` block, encloses code that can be executed to respond to particular exception.

`finally` ... follows a `try` block and (typically) one or more `catch` blocks, encloses a block of code that will execute regardless of whether any exceptions are thrown.

`throws` ... Indicates that a method may throw an exception.

`throw` ... is used to throw an exception (object).

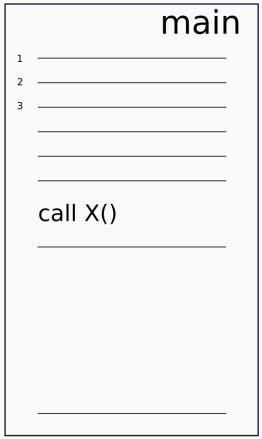
Your Questions

Next Lecture

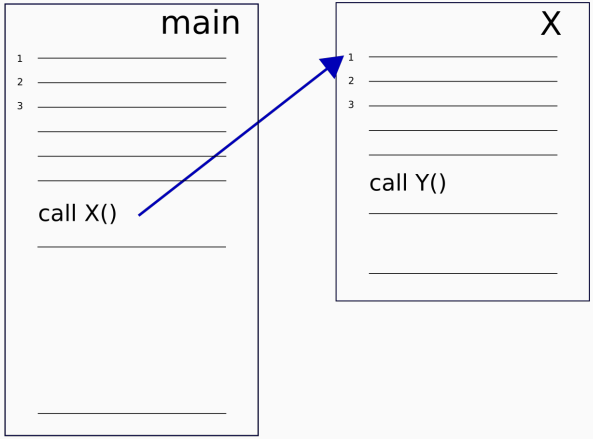
- A2 stats etc
- Writing Custom Exceptions
- Exceptions in Python

Recap Exceptions in Java

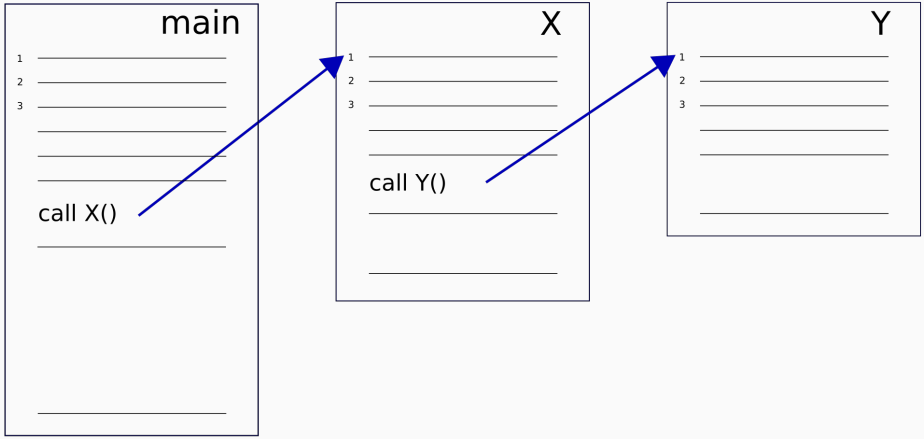
Normal Program Flow



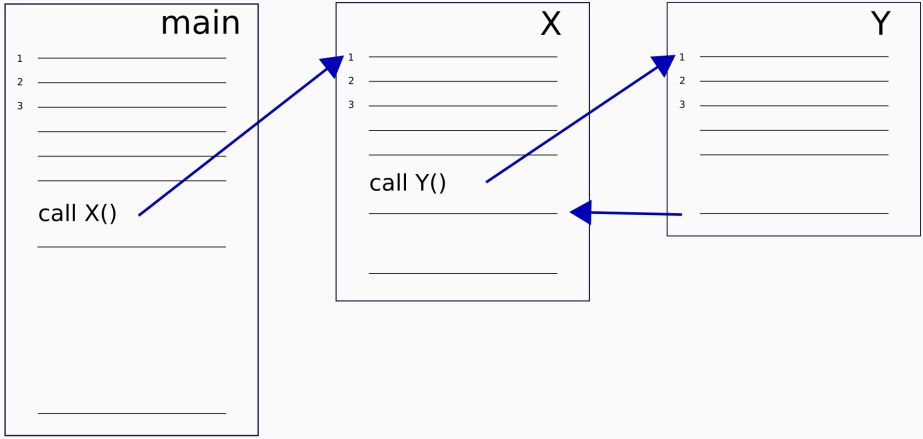
Normal Program Flow



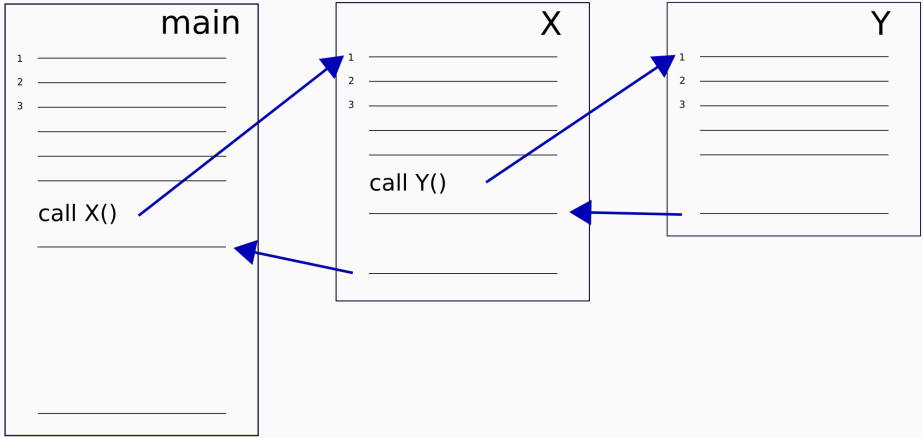
Normal Program Flow



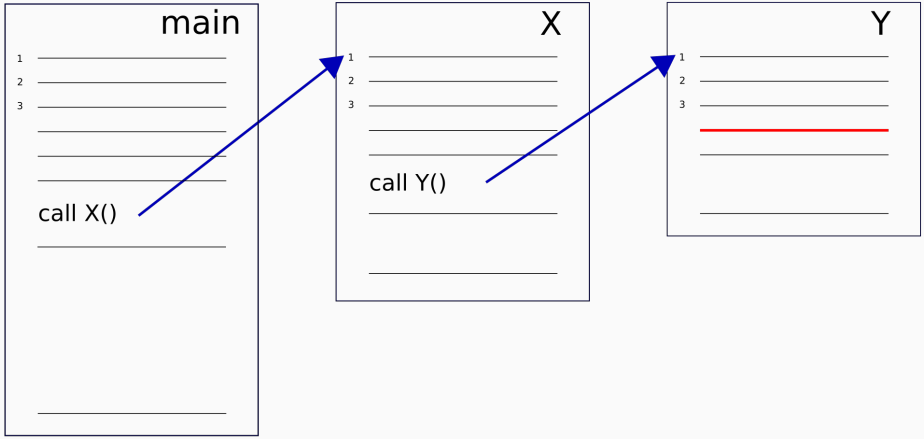
Normal Program Flow



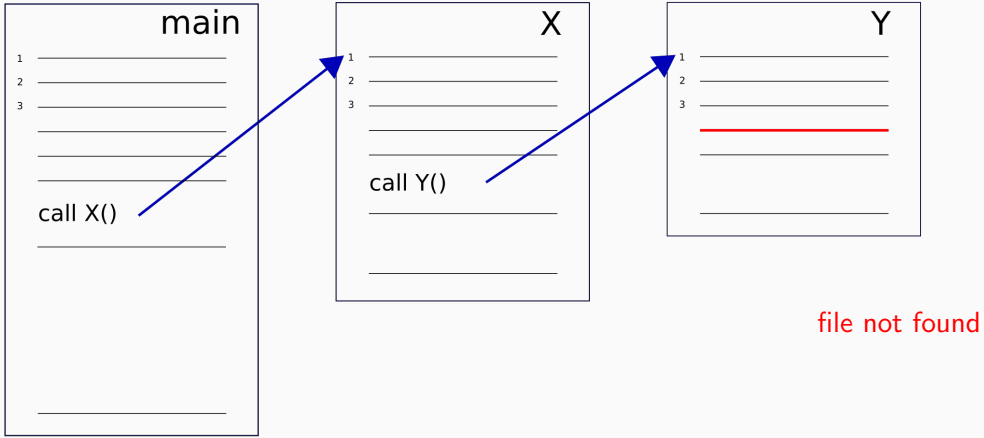
Normal Program Flow



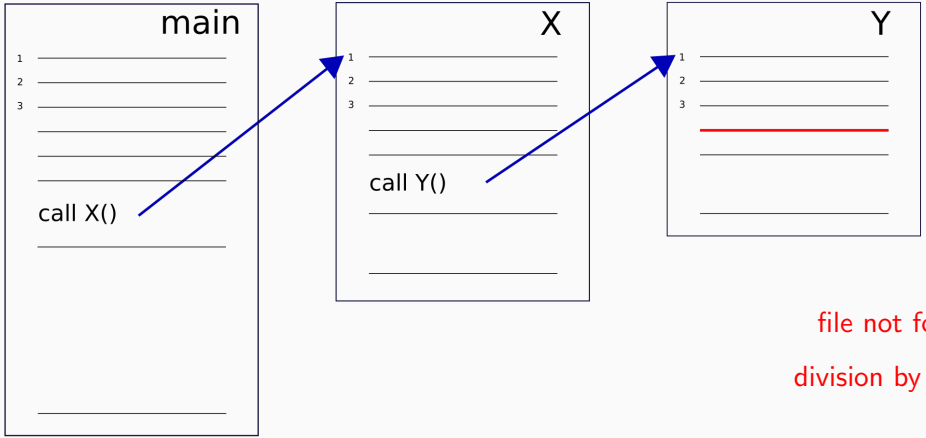
Error occurred! What next?



Error occurred! What next?



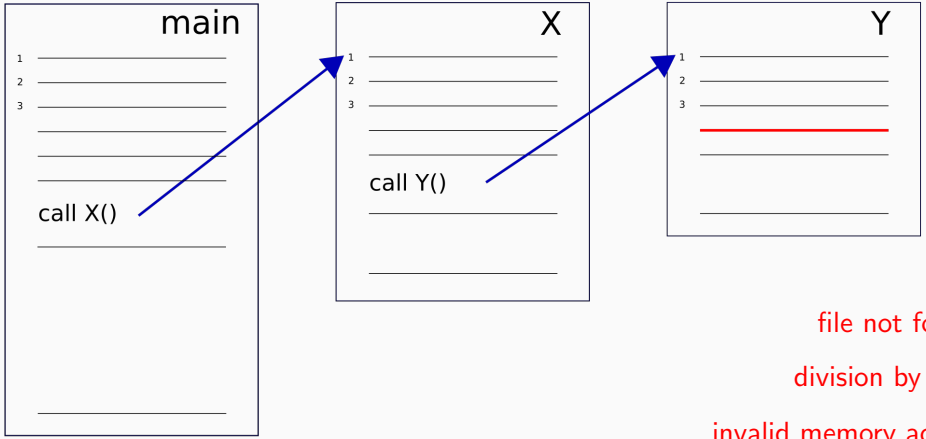
Error occurred! What next?



file not found

division by zero

Error occurred! What next?

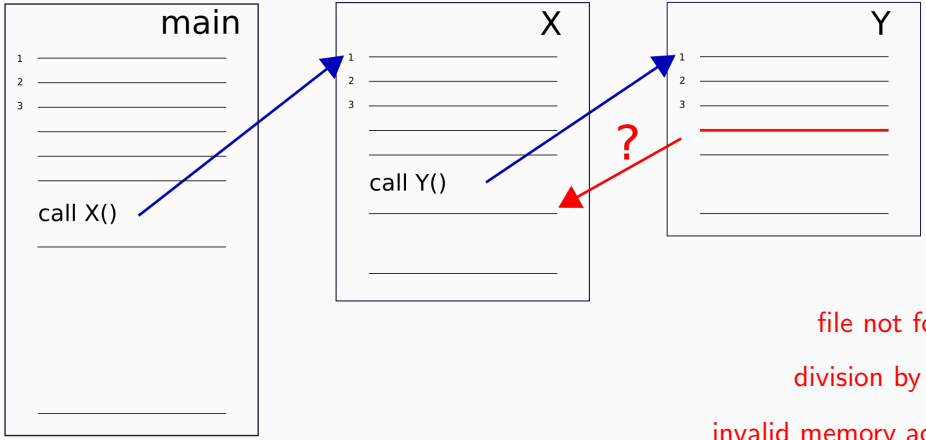


file not found

division by zero

invalid memory access

Error occurred! What next?

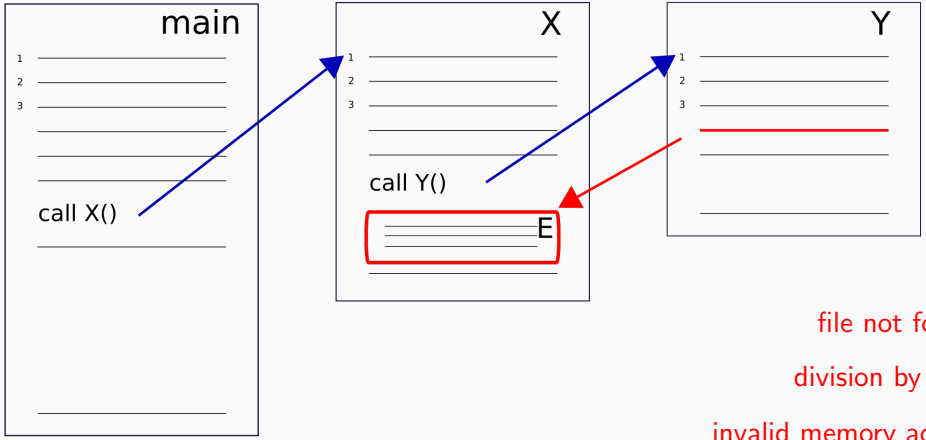


file not found

division by zero

invalid memory access

Error occurred! What next?

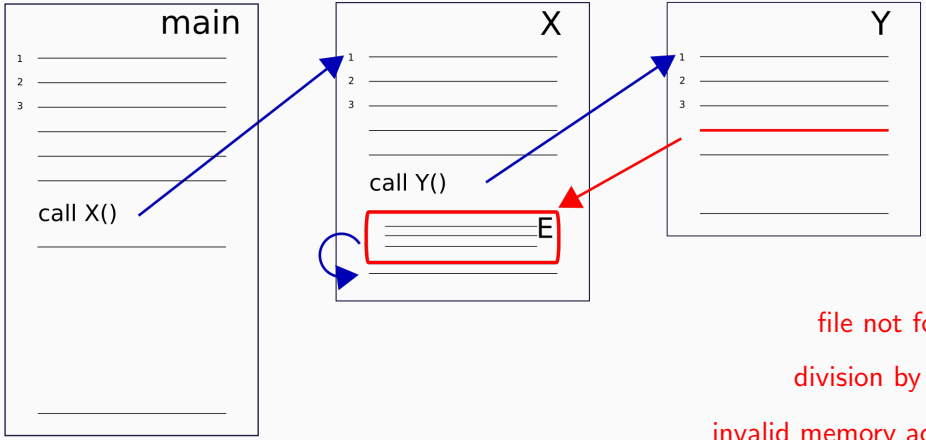


file not found

division by zero

invalid memory access

Error occurred! What next?

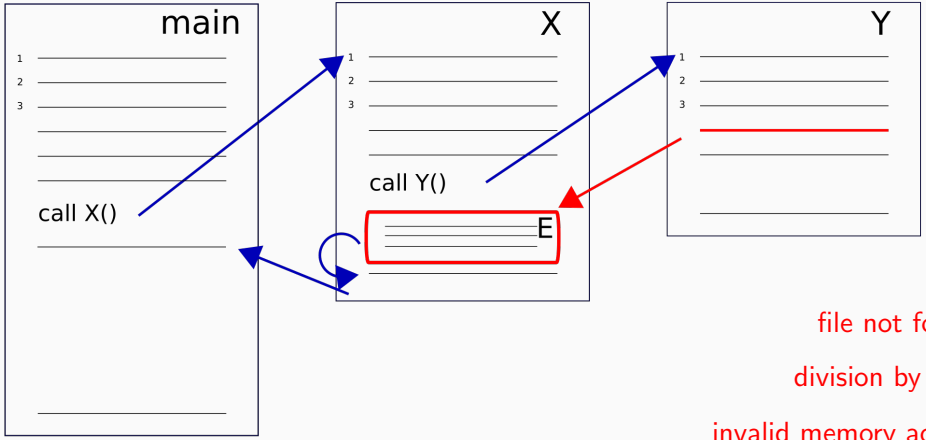


file not found

division by zero

invalid memory access

Error occurred! What next?

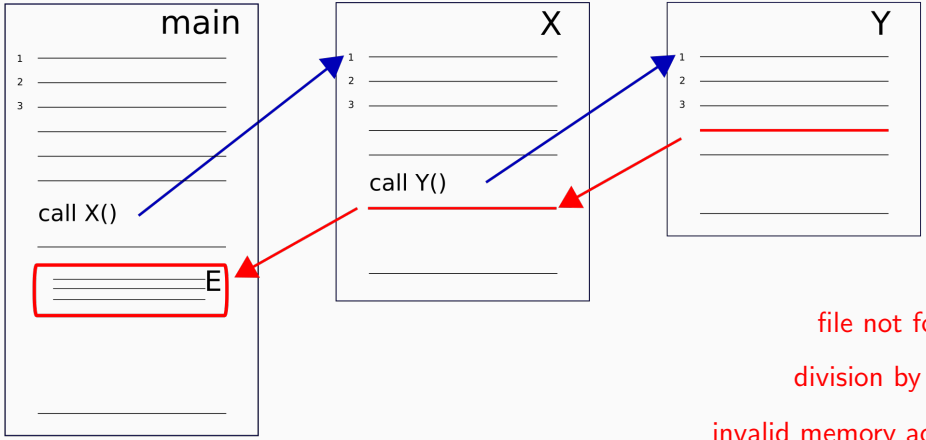


file not found

division by zero

invalid memory access

Error occurred! What next?



file not found

division by zero

invalid memory access

Exceptions are Objects!

When an anomaly occurs an `Exception` object is created.

- it represents the situation and has relevant info (as attributes)
- the environment looks for a handler based on its type
- to report an anomaly you instantiate, then throw an exception
- to deal with possible exceptions, define suitable handler

Exceptions Handling – Common Pattern

```
1 // PSEUDOCODE
2 try {
3     do stuff;       // errors may happen
4 }
5 catch exception {
6     deal with it;   // handler
7 }
```

Exceptions Handling – Common Pattern

```
1 // PSEUDOCODE
2 try {
3     do stuff;       // errors may happen
4 }
5 catch exception {
6     deal with it;   // handler
7 }
```

What should the handler do?

Exceptions Handling – Common Pattern

```
1 // PSEUDOCODE
2 try {
3     do stuff;       // errors may happen
4 }
5 catch exception {
6     deal with it;   // handler
7 }
```

What should the handler do?

report and re-throw?

Exceptions Handling – Common Pattern

```
1 // PSEUDOCODE
2 try {
3     do stuff;       // errors may happen
4 }
5 catch exception {
6     deal with it;   // handler
7 }
```

What should the handler do?

report and re-throw?

log and continue?

Exceptions Handling – Common Pattern

```
1  // PSEUDOCODE (Java-ish)
2  try {
3    // open a file for writing
4  }
5  catch (FileNotFoundException e) {
6    // complain that e.path was invalid
7  }
8  catch (FileAccessException e) {
9    // complain that file is read-only
10 }
```

Catch or Declare

Java wants you to specify as part of a methods signature, which exceptions it may throw...

Catch or Declare

Java wants you to specify as part of a methods signature, which exceptions it may throw... to allow analysis at compile time.

Catch or Declare

Java wants you to specify as part of a methods signature, which exceptions it may throw... to allow analysis at compile time.

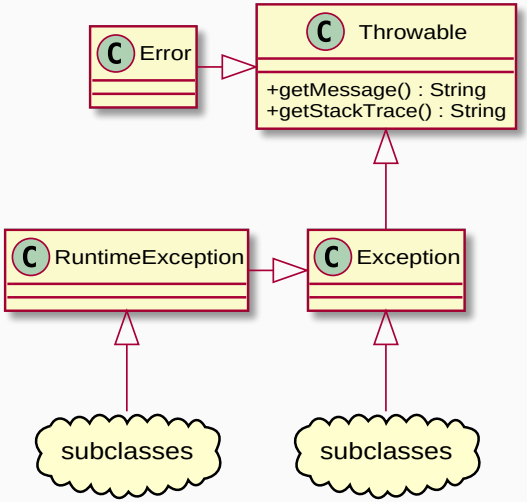
```
1  public String myMethod() throws XException {
2
3    // call some method that is declared to possibly
4    // throw XExceptions
5    someMethod();
6
7    // or explicitly create and throw one yourself
8    throw new XException("some description");
9  }
```

Catching an Exception

```
1  import java.io.IOException;
2
3  public class ReadFromKeyboard3 {
4    public static void main (String args[]) {
5      try{
6        // read one byte from stdin
7        int myChar = System.in.read();
8        // Interpret that int as a char and print
9        System.out.println((char) myChar);
10     }
11     catch(IOException e)  {
12       e.printStackTrace();
13     }
14   }
15 }
```

Throwables

`java.lang` defines the following classes that can be `throw`n



Writing your own Exceptions

All exceptions are objects of type `java.lang.Exception` so you can create your own custom exception classes by extending this class.

How you do this depends upon the type of exception you want. Is it an exception that can be checked and from which you can recover? Or is it an exception that should stop the program (i.e. is unrecoverable)?

In the first case, you want to extend `Exception`, whereas in the second case you can extend `RuntimeException`.

Before you do this, consider whether some built-in exception is sufficient for your purposes.

Some Java exceptions

- `ArithmeticException` - Arithemtical operations (e.g. dividing by zero).
- `ArrayIndexOutOfBoundsException` - Trying to access an index that is negative or bigger than the size of an array.
- `FileNotFoundException` - Trying to access a file that doesn't exist or is otherwise not accessible.
- `NullPointerException` - Referring to members of a null object.
- `RuntimeException` - Any exception that occurs during runtime.
- `StringIndexOutOfBoundsException` - Using an index that is negative or beyond the length of the `String` object.

A custom “Negative Number Exception”

```
1  /** An exception for complaining abut negative integers */
2  public class NegativeNumberException extends Exception {
3
4      public NegativeNumberException()  {}
5
6      public NegativeNumberException(String message)  {
7          super(message);
8      }
9
10     public NegativeNumberException(Throwable cause)  {
11         super(cause);
12     }
13
14     public NegativeNumberException(String message, Throwable cause)  {
15         super(message, cause);
16     }
17 }
```

A custom "Negative Number Exception"

```
1  /** An exception for complaining abut negative integers */
2  public class NegativeNumberException extends Exception {
3
4      public NegativeNumberException()  {}
5
6      public NegativeNumberException(String message)  {
7          super(message);
8      }
9
10     public NegativeNumberException(Throwable cause)  {
11         super(cause);
12     }
13
14     public NegativeNumberException(String message, Throwable cause)  {
15         super(message, cause);
16     }
17 }
```

Q: is `NegativeNumberException` checked or unchecked?

Using our custom Exception (I)

```
1  import java.util.Scanner;
2  import java.util.InputMismatchException;
3
4  public class ReadNonNegLong {
5
6     public static void main (String args[]) {
7        //  Create a Scanner object to read from "standard input".
8        Scanner scan = new Scanner(System.in);
9
10       //  Try to get a long as input.  This can throw a RuntimeException
11       try {
12         System.out.print("Enter an integer (or long): ");
13         long l = scan.nextLong();
14         if (l < 0) {
15             throw new NegativeNumberException("Sorry, your input
                   isnegative!");
16         }
17         System.out.println(l);
18       }
```

Using our custom Exception (II)

```
19          catch (InputMismatchException e)  {
20            System.out.println("Oops!  That input wasn't an integer");
21          }
22          catch (NegativeNumberException e)  {
23            System.out.println();
24            System.out.println(e);
25          }
26      }
27  }
```

Using our custom Exception (III)

```
$ java ReadNonNegLong
Enter an integer (or long): -12
NegativeNumberException: Sorry, your input is negative!

$ java ReadNonNegLong
Enter an integer (or long): five
Oops! That input wasn't an integer

$ java ReadNonNegLong
Enter an integer (or long): 343
343
```

Exceptions in Python

Exceptions in Python v Java

Python and Java's exception handling mechanism is almost the same:

- Exceptions are objects
- Exception classes form a hierarchy
- built-in syntax to throw/catch them.

Exceptions in Python v Java

Python and Java's exception handling mechanism is almost the same:

- Exceptions are objects
- Exception classes form a hierarchy
- built-in syntax to throw/catch them.

Python

- "raises" exceptions, where Java "throws" them.
- does not distinguish runtime/compile time exceptions as it is not a compiled language (let's ignore JIT etc.)

Exceptions in Python

Raising an exception in Python:

```
1  if x > 5:
2      raise Exception("x should not exceed 5!"))
```

Exceptions in Python

Raising an exception in Python:

```
if x > 5:
    raise Exception("x should not exceed 5!"))
```

Catching an exception in Python:

```
try:
  result = 7 / 0
except ZeroDivisionError as error:
  print(error)  # error is the exception object
```

Exceptions in Python v Java

Java

```java
try {
    // dangerous code
}
catch (XException e) {
  // e is the exception object
  System.out.println(e)
}
catch (YException e) {
  // handle Y
}
finally {
  // is always called
}
```

Python

```python
try:
  # dangerous code
except XException as e:
  # e is the exception object
  print(e)
except YException as e:
  # e is the exception object
  print(e)
else:
  # is called if try was
      successful
finally:
  # is always called
```

Summary

We looked at...

- Exceptions in general (OOP)
- Exceptions in Java
  - Exception, Error, Throwable
  - Checked vs. Unchecked
  - Custom Exceptions
- Exceptions in Python

Next Week

- Input/Output
- Streams
- Regular Expressions