# Fundamentos de programação

Tratamento de exceções

Edson Moreno

edson.moreno@pucrs.br

http://www.inf.pucrs.br/~emoreno

# Exception Handling

- There are two aspects to dealing with run-time program errors:
  1) Detecting Errors

     This is the easy part. You can 'throw' an exception

     Use the throw statement to signal an exception

     ```
     if (amount > balance)
     {
       // Now what?
     }
     ```

  2) Handling Errors

     This is more complex. You need to 'catch' each possible exception and react to it appropriately

- Handling recoverable errors can be done:
  - Simply: exit the program
  - User-friendly: As the user to correct the error

# Throwing an Exception

- When you throw an exception, you are throwing an object of an exception class
  - Choose wisely!
  - You can also pass a descriptive String to most exception objects

```
if (amount > balance)
{
   throw new IllegalArgumentException("Amount exceeds balance");
}
balance = balance - amount;
```

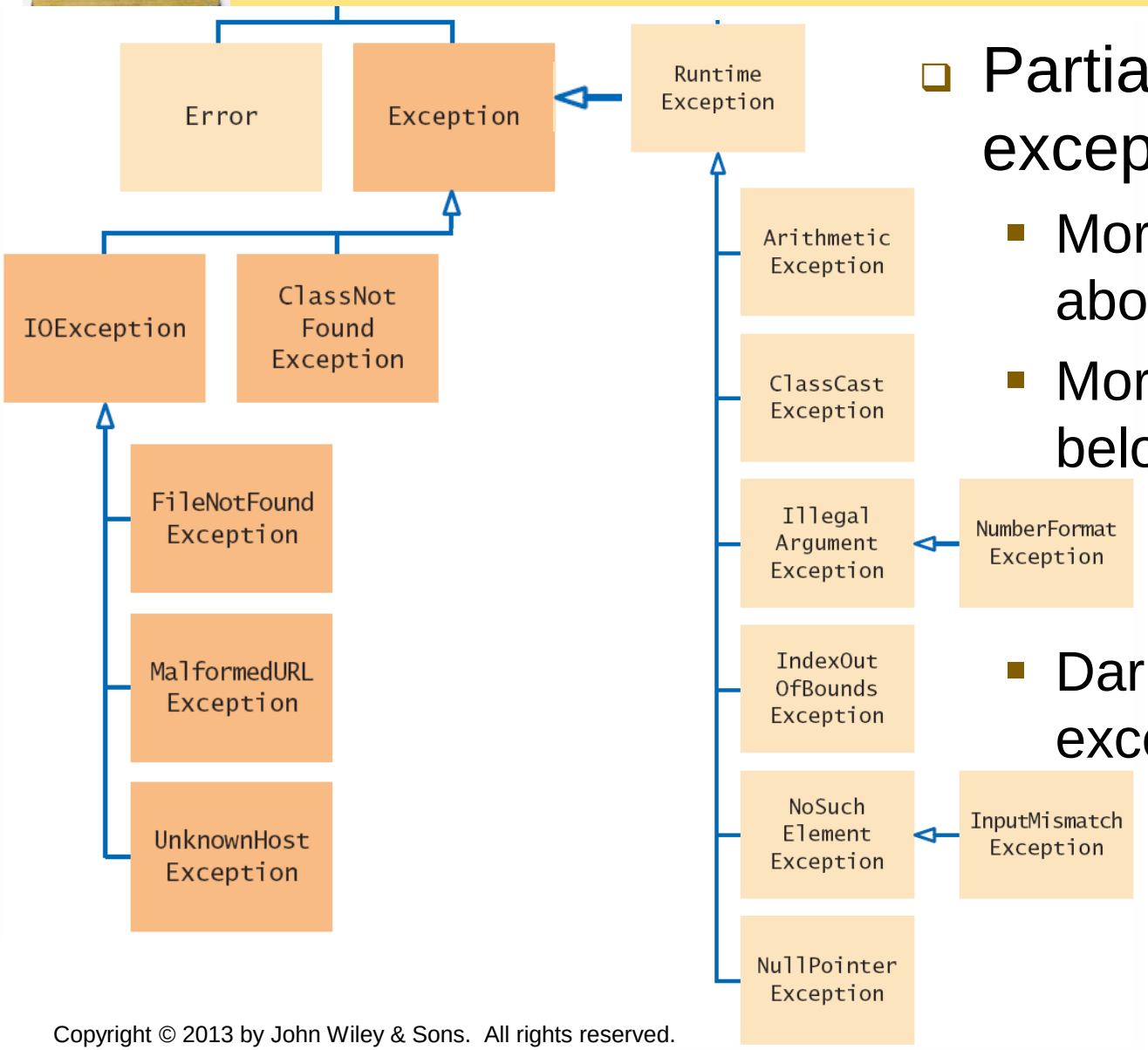
Most exception objects can be constructed with an error message.

A new exception object is constructed, then thrown.

This line is not executed when the exception is thrown.

When you throw an exception, the normal control flow is terminated.

# Exception Classes




- Partial hierarchy of exception classes
  - More general are above
  - More specific are below
  - Darker are Checked exceptions

# Catching Exceptions

- Exceptions that are thrown must be 'caught' somewhere in your program

```
try
{
   String filename = . . .;
   Scanner in = new Scanner(new File(filename));
   String input = in.next();
   int value = Integer.parseInt(input);
   . . .
}
catch (IOException exception)
{
   exception.printStackTrace();
}
catch (NumberFormatException exception)
{
   System.out.println("Input was not a number");
}
```

Surround method calls that can throw exceptions with a 'try block'.

FileNotFoundException

NoSuchElementException

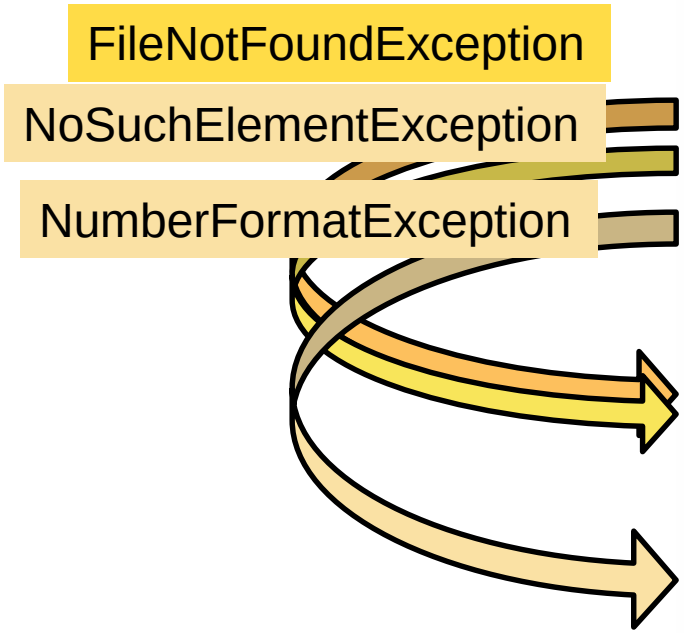
NumberFormatException

Write 'catch blocks' for each possible exception.

It is customary to name the exception parameter either 'e' or 'exception' in the catch block.

# Catching Exceptions

- When an exception is detected, execution 'jumps' immediately to the first matching `catch` block
  - `IOException` matches both `FileNotFoundException` and `NoSuchElementException` is not caught




```
try
{
   String filename = . . .;
   Scanner in = new Scanner(new File(filename));
   String input = in.next();
   int value = Integer.parseInt(input);
   . . .
}
catch (IOException exception)
{
   exception.printStackTrace();
}
catch (NumberFormatException exception)
{
   System.out.println("Input was not a number");
}
```

# Catching Exceptions

This constructor can throw a FileNotFoundException.

```
try
{
   Scanner in = new Scanner(new File("input.txt"));
   String input = in.next();
   process(input);
}
catch (IOException exception)
{
   System.out.println("Could not open input file");
}
catch (Exception except)
{
   System.out.println(except.getMessage);
}
```

When an IOException is thrown, execution resumes here.

This is the exception that was thrown.

Additional catch clauses can appear here. Place more specific exceptions before more general ones.

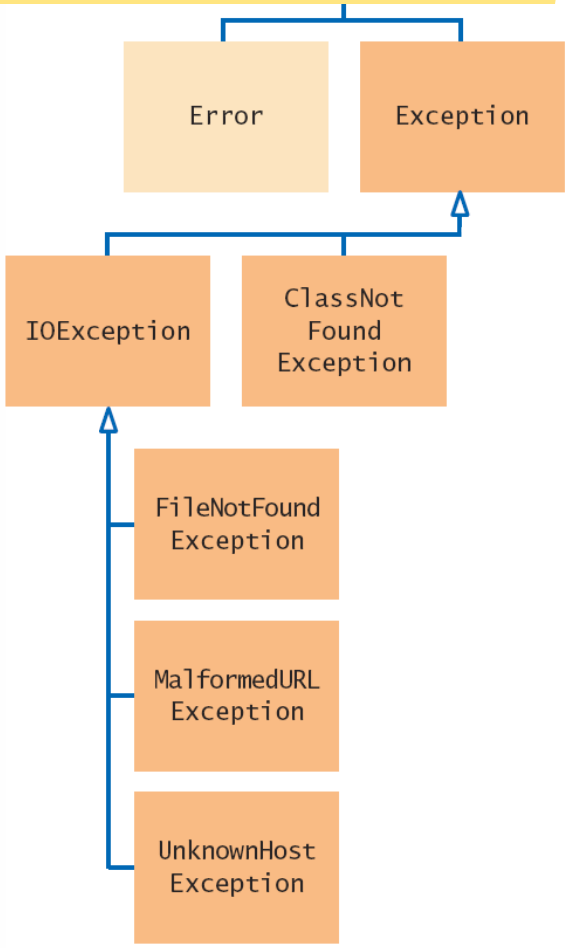
A FileNotFoundException is a special case of an IOException.

- Some exception handling options:
  - Simply inform the user what is wrong
  - Give the user another chance to correct an input error
  - Print a 'stack trace' showing the list of methods called

```
exception.printStackTrace();
```

# Checked Exceptions

- Throw/catch applies to three types of exceptions:
  - **Error:** Internal Errors
    - not considered here
  - **Unchecked:** RunTime Exceptions
    - Caused by the programmer
    - Compiler **does not check** how you handle them
  - **Checked:** All other exceptions
    - Not the programmer's fault
    - Compiler **checks** to make sure you handle these
    - Shown darker in Exception Classes




Checked exceptions are due to circumstances that the programmer cannot prevent.

# The `throws` Clause

- Methods that use other methods that may `throw` exceptions must be declared as such
  - Declare all **checked** exceptions a method throws
  - You may also list **unchecked** exceptions

```
public static String readData(String filename)
      throws FileNotFoundException, NumberFormatException
```

You must specify all checked exceptions that this method may throw.

You may also list unchecked exceptions.

# The `throws` Clause (continued)

- If a method handles a checked exception internally, it will no longer `throw` the exception.
  - The method does not need to declare it in the `throws` clause
- Declaring exceptions in the `throws` clause ‘passes the buck’ to the calling method to handle it or pass it along.

# The `finally` clause

- `finally` is an optional clause in a `try/catch` block
  - Used when you need to take some action in a method whether an exception is thrown or not.
    - The finally block is executed in both cases
  - Example: Close a file in a method in all cases

```
public void printOutput(String filename) throws IOException
{
   PrintWriter out = new PrintWriter(filename);
   try
   {
      writeData(out);   // Method may throw an I/O Exception
   }
   finally
   {
      out.close();
   }
}
```

Once a `try` block is entered, the statements in a `finally` clause are guaranteed to be executed, whether or not an exception is thrown.

# The `finally` Clause

- Code in the `finally` block is always executed once the `try` block has been entered

This variable must be declared outside the `try` block so that the `finally` clause can access it.

```
PrintWriter out = new PrintWriter(filename);
try
{
   writeData(out);
}
finally
{
   out.close();
}
```

This code may throw exceptions.

This code is always executed, even if an exception occurs.

# Programming Tip 7.1

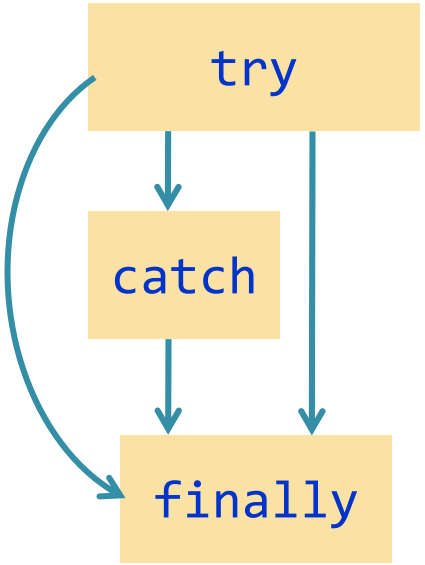
- Throw Early
  - When a method detects a problem that it cannot solve, it is better to throw an exception rather than try to come up with an imperfect fix.
- Catch Late
  - On the other hand, a method should only catch an exception if it can really remedy the situation.
  - Otherwise, the best remedy is simply to have the exception propagate to its caller, allowing it to be caught by a competent handler.

# Programming Tip 7.2

- Do Not Hide Exceptions
  - When you call a method that throws a checked exception and you haven't specified a handler, the compiler complains.
  - It is tempting to write a 'do-nothing' catch block to 'hide' the compiler and come back to the code later. **Bad Idea!**
    - Exceptions were designed to transmit problem reports to a competent handler.
    - Installing an incompetent handler simply hides an error condition that could be serious..

# Programming Tip 7.3

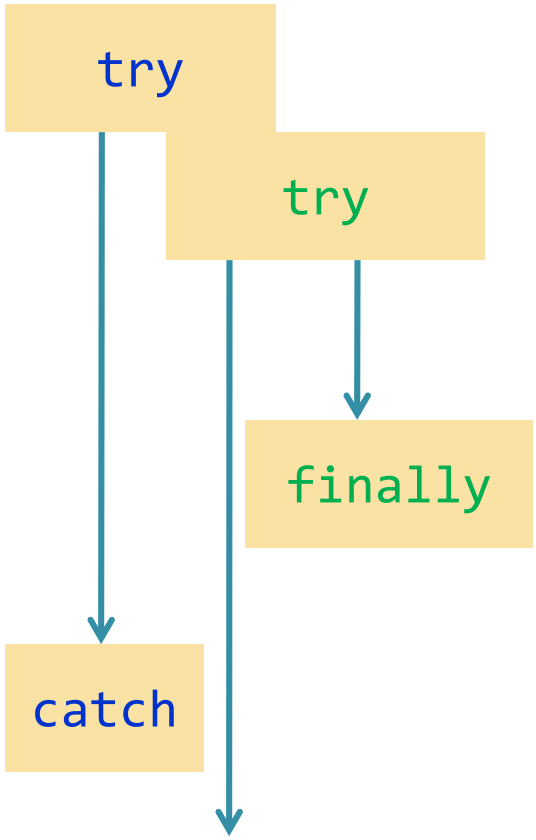
- Do not use `catch` and `finally` in the same `try` block
  - The `finally` clause is executed whenever the try block is exited in any of three ways:
    1. After completing the last statement of the `try` block
    2. After completing the last statement of a `catch` clause, if this try block caught an exception
    3. When an exception was thrown in the `try` block and not caught



# Programming Tip 7.3

- It is better to use two (nested) `try` clauses to control the flow

```
try
{
   PrintWriter out = new PrintWriter(filename);
   try
   {       // Write output    }
   finally
   {  out.close(); }  // Close resources
}
catch (IOException exception)
{
   // Handle exception
}
```



# Handling Input Errors

- File Reading Application Example
  - Goal: Read a file of data values
    - First line is the count of values
    - Remaining lines have values
  - Risks:
    - The file may not exist
      - `Scanner` constructor will throw an exception
      - `FileNotFoundException`
    - The file may have data in the wrong format
      - Doesn't start with a count
        - » `NoSuchElementException`
      - Too many items (count is too low)
        - » `IOException`

```
3
1.45
-2.1
0.05
```

# Handling Input Errors: `main`

- Outline for method with all exception handling

```
boolean done = false;
while (!done)
{
  try
  {
    // Prompt user for file name
    double[] data = readFile(filename);   // May throw exceptions
    // Process data
    done = true;
  }
  catch (FileNotFoundException exception)
  {      System.out.println("File not found.");  }
  catch (NoSuchElementException exception)
  {      System.out.println("File contents invalid.");  }
  catch (IOException exception)
  {      exception.printStackTrace();  }
}
```

# Handling Input Errors: `readFile`

- Calls the `Scanner` constructor
- No exception handling (no `catch` clauses)
- `finally` clause closes file in all cases (exception or not)
- `throws IOException` (back to `main`)

```
public static double[] readFile(String filename) throws IOException
{
   File inFile = new File(filename);
   Scanner in = new Scanner(inFile);
   try
   {
      return readData(in);   // May throw exceptions
   }
   finally
   {
      in.close();
   }
}
```

# Handling Input Errors: readData

- No exception handling (no `try` or `catch` clauses)
- `throw` creates an `IOException` object and exits
- unchecked `NoSuchElementException` can occur

```
public static double[] readData(Scanner in) throws IOException
{
   int numberOfValues = in.nextInt();   // NoSuchElementException
   double[] data = new double[numberOfValues];
   for (int i = 0; i < numberOfValues; i++)
   {
      data[i] = in.nextDouble();         // NoSuchElementException
   }
   if (in.hasNext())
   {
      throw new IOException("End of file expected");
   }
   return data;
}
```

# Exercício

- Crie um programa, o qual deverá ter um método chamado getInt. O método deve solicitor que o usuário entre com um valor inteiro. Capture este valor, e caso ele não seja um valor inteiro (e.g. string/double), lance uma exceção do tipo IllegalArgumentException; Caso o valor esteja ok, retorne o valor inteiro.

# Exercício

- Modifique o programa anterior de tal forma que o método getInt lance uma exceção do tipo IOException ao invés de IllegalArgumentException. Modifique o corpo principal do programa (i.e. a main ) de tal forma que ela capture a exceção e imprima a exceção IOException.