(remedial) Java

anastasia.bezerianos@lri.fr

Jars

Disk organization of Packages

- Packages are just directories
- For example
- class3.inheritanceRPG is located in
- <workspace>\RemedialJava\src\class3\inheritenceRPG
- The compiled classes under …\RemedialJava\bin\...

Jar

- A JAR file (Java ARchive) is a collection of classes and metadata to distribute applications and libraries in Java
- In eclipse, to add jars of others go to Project➔Properties➔Build Path

Jar

- JARs can be executable if they include a class with a function `main()`, and when we create them we say where it can be found
- In eclipse:
  - create a run configuration, chose the main class
  - in the project export, choosing the run configuration. Make sure to add sources. Chose your export location + name
  - in a terminal run> java –jar myJarName.jar (no GUI …)

Jar

- As JARs become more complex (many packages, resources like images, etc.) you need to test them to make sure things are exported properly. Classic example image paths (several tutorials online)
- In classes, unless otherwise specified, hand in an archive of your project, and make sure all paths for reading files (later) are relative

Exercise 1

www.lri.fr/~anab/teaching/remedial-java/ex-class45.pdf

Reminder:

- Create a run configuration (Run→Run Configuration) and chose the correct class with the main you want
- Project→Export jar File, make sure to add sources.
- Project→Export runnable jar

- Can run a runnable jar in a console using:

```
java –jar myJarName.jar
```


Exceptions

Exceptions

NullPointerException

ArrayIndexOutOfBoundsException

FileNotFoundException

ClassCastException

RuntimeException

Exceptions

An exception is an event triggered when something unexpected happens, e.g.,

```
null.someMethod();

(new int[1])[1] = 0;

int i = “string”;
```

Why use Exceptions?

- To tell the code using your method that something went wrong
- Debugging and understanding control flow

```
public class ExceptionExample {

    public static String getArgument(String[] args, int index){
        return args[index];
    }

    public static void main(String[] args){
        getArgument(args,0);
    }
}
```


Why use Exceptions?

- To tell the code using your method that something went wrong
- Debugging and understanding control flow

Exception in thread "main" java.lang.**ArrayIndexOutOfBoundsException: 0**
at ExceptionExample.main(RuntimeException.java:10)

Accessed index 0, which isn't in the array.
The method that called it was main at line 10

Why use Exceptions?

- Problem in C:
  - generally we assign a return value that signals errors
  - then we propagate errors manually
- In Java (and C++) the mechanism to propagate errors is managed by the language

How do exceptions happen?

- Java doesn't know what to do, so it:
  - Creates an Exception object
  - Includes some useful information
  - "throws" the Exception

How do exceptions happen?

- Java doesn't know what to do, so it:
  - Creates an Exception object
  - Includes some useful information
  - "throws" the Exception

- You can (create and) throw Exceptions too!
  - Exception is a class, so you can create your own, but there are many already (see java documentation)
  - For now let us see how we can throw exceptions

Throwing exceptions

```
public static String getArgument (String[] args, int index)
 throws IlligalArgumentException {

   if ( args.length == 0 )
      throw new IllegaleArgumentException("Array is empty!");
   if ( index < 0 || index >= args.length )
      throw new IllegaleArgumentException("bad index " + index);
   return args[index];
}
```

- **throws** tells Java that getArgument() may throw the IlligalArgumentException
- **throw** actually throws the Exception
- You can throw more than one …

Catching exceptions

- If no-one catches the exception,
  - Java will print an error message, but customized by your throw

```
Exception in thread "main" java.lang.IllegalArgumentException: Array empty!
    at ExceptionExample1.getArgument(ExceptionExample1.java:8)
    at ExceptionExample1.main(ExceptionExample1.java:28)
```

- … or we can catch it and deal with it

Catching exceptions

- Let's complicate things a bit

```
public class ExceptionExample {

   public static String getArgument (String[] args, int index)
    throws IlligalArgumentException {

      if ( args.length == 0 )
         throw new IllegaleArgumentException("Array is empty!");
      if ( index < 0 || index >= args.length )
         throw new IllegaleArgumentException("bad index " + index);
      return args[index];
   }
   public static String justInTheWay(String[] args, int index){
      return getArgument(args, index);
   }
   public static void main(String[] args){
      justInTheWay(args,0);
   }
}
```

Catching exceptions

- When a method throws an exception, Java now expects someone to deal with the exception, by:
  - Catching it, or
  - Rethrowing it

Catching exceptions

What it does:

- try to run the code that may throw an exception
- tell Java what to do if it sees the exception (catch)

```
public static String justInTheWay (String[] args, int index) {
    String answer = "";
    try {
        answer = getArgument(args,index);
    } catch ( IllegalArgumentException e){
        System.out.println("Oh no …"); // continue gracefully
    }
    return answer;
}
```

Rethrowing exceptions

Or, if you don't know what to do with it …

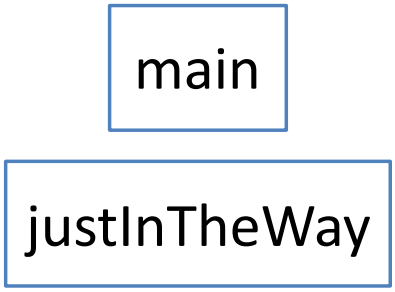
- rethrow it to the method that called you …

```
public static String justInTheWay (String[] args, int index)
  throws IllegalArgumentException {
       getArgument(args,index);
}
```

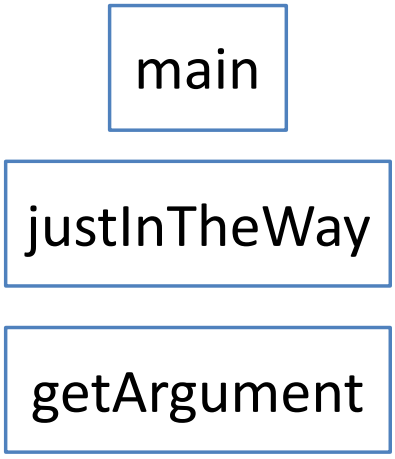

Rethrowing exceptions

main

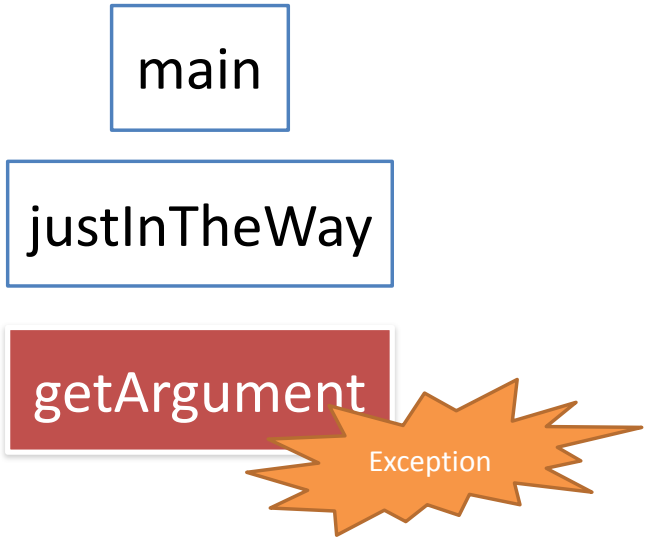
Rethrowing exceptions



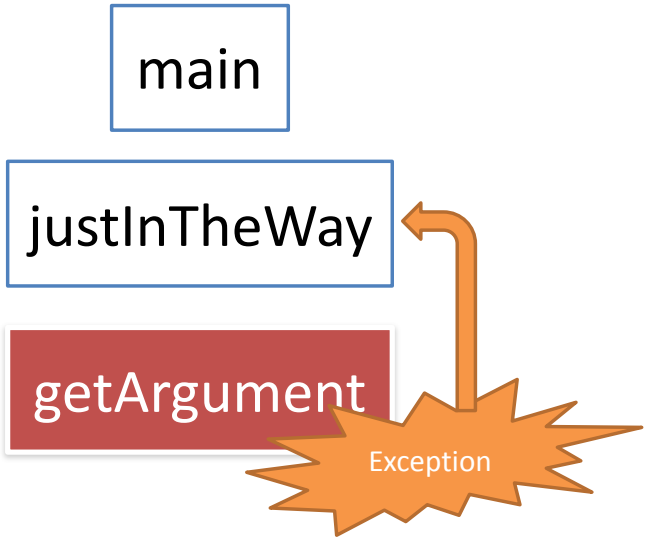
Rethrowing exceptions



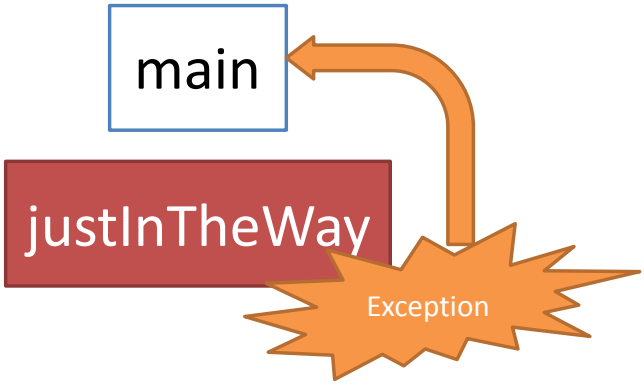
Rethrowing exceptions



Rethrowing exceptions



Rethrowing exceptions



Rethrowing exceptions

- If no-one catches it, …. you get your error message and exception …

```
Exception in thread "main" java.lang.IllegalArgumentException: Array empty!
        at class4.exceptionExamples.ExceptionExample1.getArgument(ExceptionExample1.java:8)
        at class4.exceptionExamples.ExceptionExample1.justInTheWay(ExceptionExample1.java:16)
        at class4.exceptionExamples.ExceptionExample1.main(ExceptionExample1.java:28)
```

Catching exceptions

Sometimes we may need to catch multiple exceptions, the order is important!

```
public static String justInTheWay (String[] args, int index)
    String answer = "";
    try {
        answer = getArgument(args,index);
        return answer;
    } catch ( IllegalArgumentException e){
        System.out.println("Wrong argument");
    } catch ( ArrayOutOfBoundsException e){
        System.out.println("Index too big for array size");
    } catch ( Exception e){ // most general Exception class
        System.out.println("Unknown error");
        e.printStackTrace(); // else risk debugging for hours …
    } finally {
        return answer; // this will happen even with exceptions
    }
}
```


Catching exceptions

Sometimes we may need to catch multiple exceptions, the order is important!

```
public static String justInTheWay (String[] args, int index)
    String answer = "";
    try {
        answer = getArgument(args,index);
        return answer;
    } catch ( IllegalArgumentException e){
        System.out.println("Wrong argument");
    } catch ( Exception e){
        System.out.println("Unknown error");
        e.printStackTrace();
    } catch ( ArrayOutOfBoundsException e){
        System.out.println("Index too big"); // never runs
    } finally {
        return answer;
    }
}
```

Catching exceptions

- Avoid treating every exception unless it is necessary (e.g., for some reason your program should continue to run even with the exceptions)

- The exception `Exception e` is the most general exception in Java (parent of all others).
  - **IF** you decide to catch it, use `e.printStackTrace()`, else you risk never actually seeing the error

Exception advantages

- Separate error handling code from normal code

- Propagate errors up the call stack to methods that know how to treat them

- Group of different error types

Don’t overdo-it

- Exceptions are there to make your error handling code easier (e.g., centralized).
- They should handle exceptional cases (that you cannot handle in your program).

e.g., imagine you have a UI interface for inputing a date. A user types “66/3/2015” or “a/01/2b”, should you include an IlligalArgumentException?

More on exceptions

https://docs.oracle.com/javase/tutorial/essential/exceptions/

http://en.wikipedia.org/wiki/Exceptions

Exercise 2

www.lri.fr/~anab/teaching/remedialJava/ex-class45.pdf

Reminder:

- Syntax

  try{ … }

  catch (EXEPTIONTYPE1 e){ … }

  catch (EXEPTIONTYPE2 e){ … }
- Can see the trace with `e.printStackTrace();`
- Do unit testing of your code without exceptions first, and parameters that violate it. Then catch the exceptions that make sense.