Lecture 4:
Exceptions.
I/O

Outline

- Access control. Class scope
- Exceptions
- I/O

Access Control

```
public class CreditCard {
   String cardNumber;
   double expenses;
   void charge(double amount) {
      expenses = expenses + amount;
   }
   String getCardNumber(String password) {
      if (password.equals("SECRET!3*!")) {
         return cardNumber;
      }
         return "jerkface";
   }
}
```

Mr. MeanGuy

```
public class Malicious {
   public static void main(String[] args) {
      maliciousMethod(new CreditCard());
   }
   static void maliciousMethod(CreditCard card)
   {
      card.expenses = 0;
      System.out.println(card.cardNumber);
   }
}
```

Public vs. Private

- Public: others can use this
- Private: only the class can use this

public/private applies to any
field or **method**

Access Control

```
public class CreditCard {
   String cardNumber;
   double expenses;
   void charge(double amount) {
      expenses = expenses + amount;
   }
   String getCardNumber(String password) {
      if (password.equals("SECRET!3*!")) {
         return cardNumber;
      }
         return "jerkface";
   }
}
```

Access Control DONE RIGHT

```
public class CreditCard {
   private String cardNumber;
   private double expenses;
   public void charge(double amount) {
      expenses = expenses + amount;
   }
   public String getCardNumber(String password)
   {
      if (password.equals("SECRET!3*!")) {
         return cardNumber;
      }
      return "jerkface";
   }
}
```

Why Access Control

- Protect private information (sorta)
- Clarify how others should use your class
- Keep implementation separate from interface

Scope

Just like methods, variables are accessible inside {}

- Method-level scope

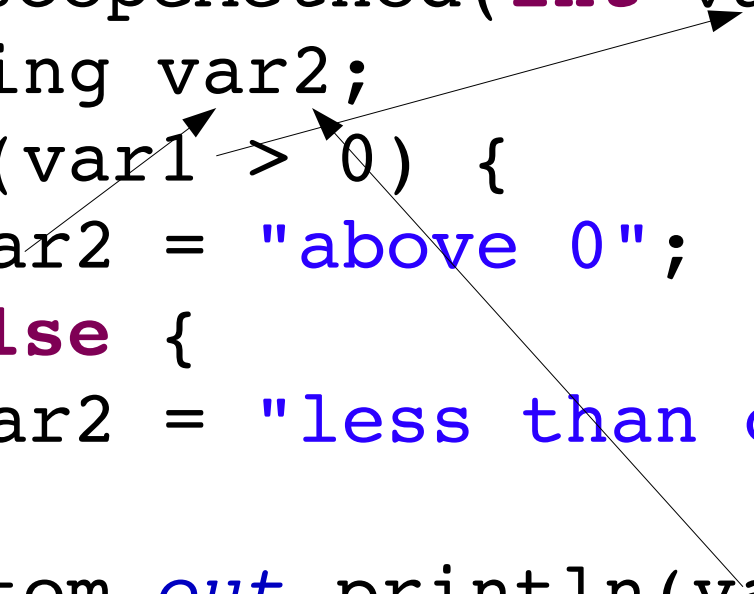
```
void method(int arg1) {
    int arg2 = arg1 + 1;
}
```

- Class-level scope

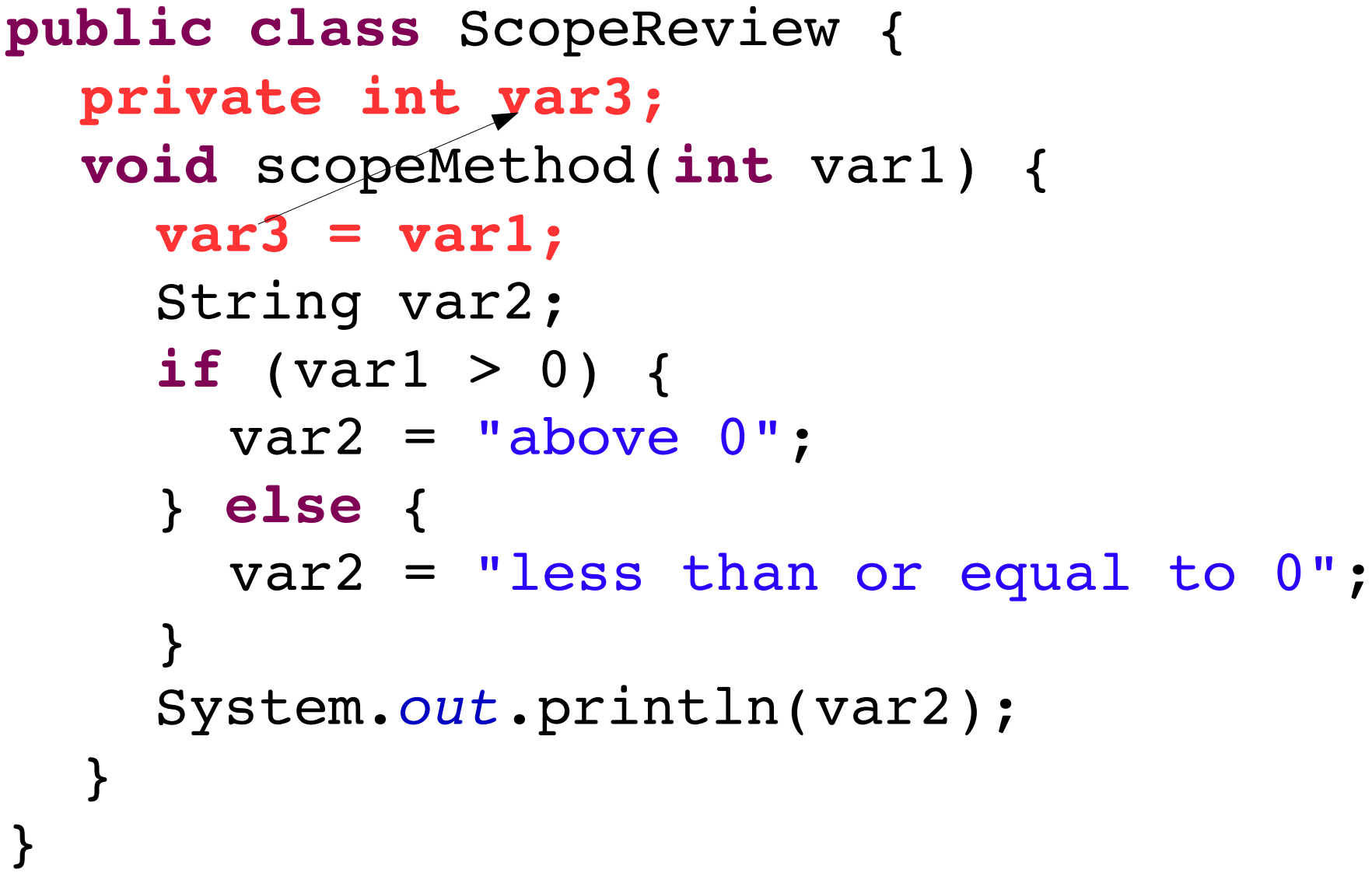
```
class Example {
    int memberVariable;
    void setVariable(int newVal) {
        memberVariable += newVal;
    }
}
```

Method Scope

```
public class ScopeReview {
   void scopeMethod(int var1) {
      String var2;
      if (var1 > 0) {
         var2 = "above 0";
      } else {
         var2 = "less than or equal to 0";
      }
      System.out.println(var2);
   }
}
```



Class Scope



```
public class ScopeReview {
   private int var3;
   void scopeMethod(int var1) {
      var3 = var1;
      String var2;
      if (var1 > 0) {
         var2 = "above 0";
      } else {
         var2 = "less than or equal to 0";
      }
      System.out.println(var2);
   }
}
```

'this' keyword

- Clarifies scope
- Means 'my object'

Usage:

```
class Example {
        int memberVariable;
        void setVariable(int newVal) {
            this.memberVariable += newVal;
        }
    }
```

Only method-level 'servings' is updated

```
public class Baby {
  int servings;
  void feed(int servings) {
    servings = servings + servings;
  }
  void poop() {
    System.out.println("All better!");
    servings = 0;
  }
}
```

Object-level 'servings' is updated

```
public class Baby {
  int servings;
  void feed(int servings) {
    this.servings =
      this.servings + servings;
  }
  void poop() {
    System.out.println("All better!");
    servings = 0;
  }
}
```

Outline

- Access control. Class scope
- Exceptions
- I/O

Exceptions

- NullPointerException
- ArrayIndexOutOfBoundsException
- ClassCastException
- RuntimeException

What is an “Exception”?

- Event that occurs when something “unexpected” happens
  - null.someMethod();
  - (new int[1])[1] = 0;
  - int i = “string”;

Why use an Exception?

- To tell the code using your method that something went wrong

```
Exception in thread "main"
  java.lang.ArrayIndexOutOfBoundsException: 5
  at RuntimeException.main(RuntimeException.java:8)
```

Accessed index 5, which isn't in the array

The method that called it was main

- Debugging and understanding control flow

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!

public class Exception

- Exception is a class
- Just inherit from it!

```
public class MyException extends Exception
{
}
```

- Or use existing ones
  - http://rymden.nu/exceptions.html

Warn Java about the Exception

```
public Object get(int index) throws
  ArrayOutOfBoundsException {
   If (index < 0 || index >= size())
      throw new
        ArrayOutOfBoundsException(“”+index);
}
```

- **`throws`** tells Java that `get` may throw the ArrayOutOfBoundsException
- **`throw`** actually throws the Exception (sorry)

Catching an Exception

- Java now expects code that calls `get` to deal with the exception by
  - Catching it
  - Rethrowing it

Catching it

- What it does
  - `try` to run some code that may throw an exception
  - Tell Java what to do if it sees the exception (`catch`)

```
try {
  get(-1);
} catch (ArrayOutOfBoundsException err) {
  System.out.println("oh dear!");
}
```

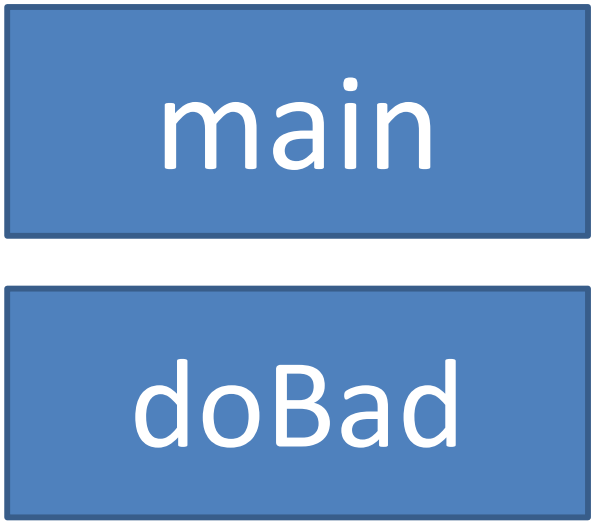
Rethrowing it

- Maybe you don’t want to deal with the Exception
- Tell Java that your method throws it too

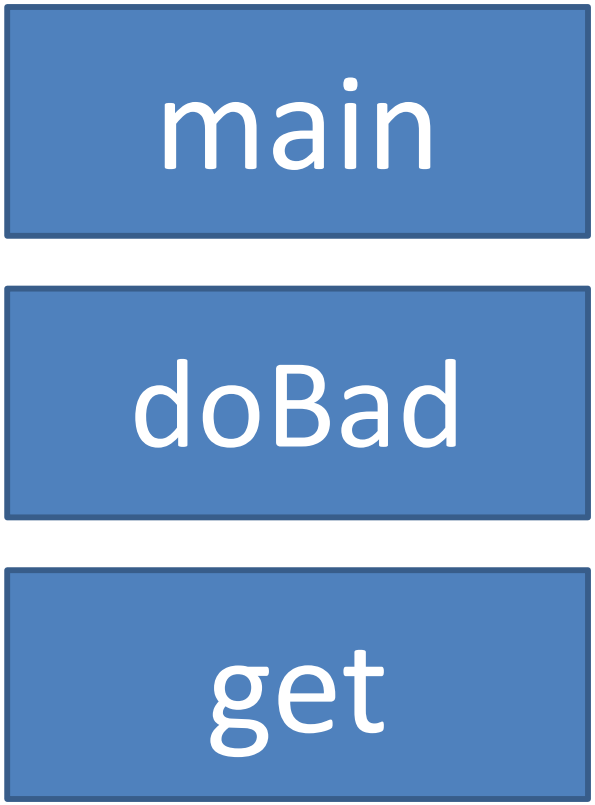
```
void doBad() throws ArrayOutOfBoundsException {
    get(-1);
}
```

Rethrowing it

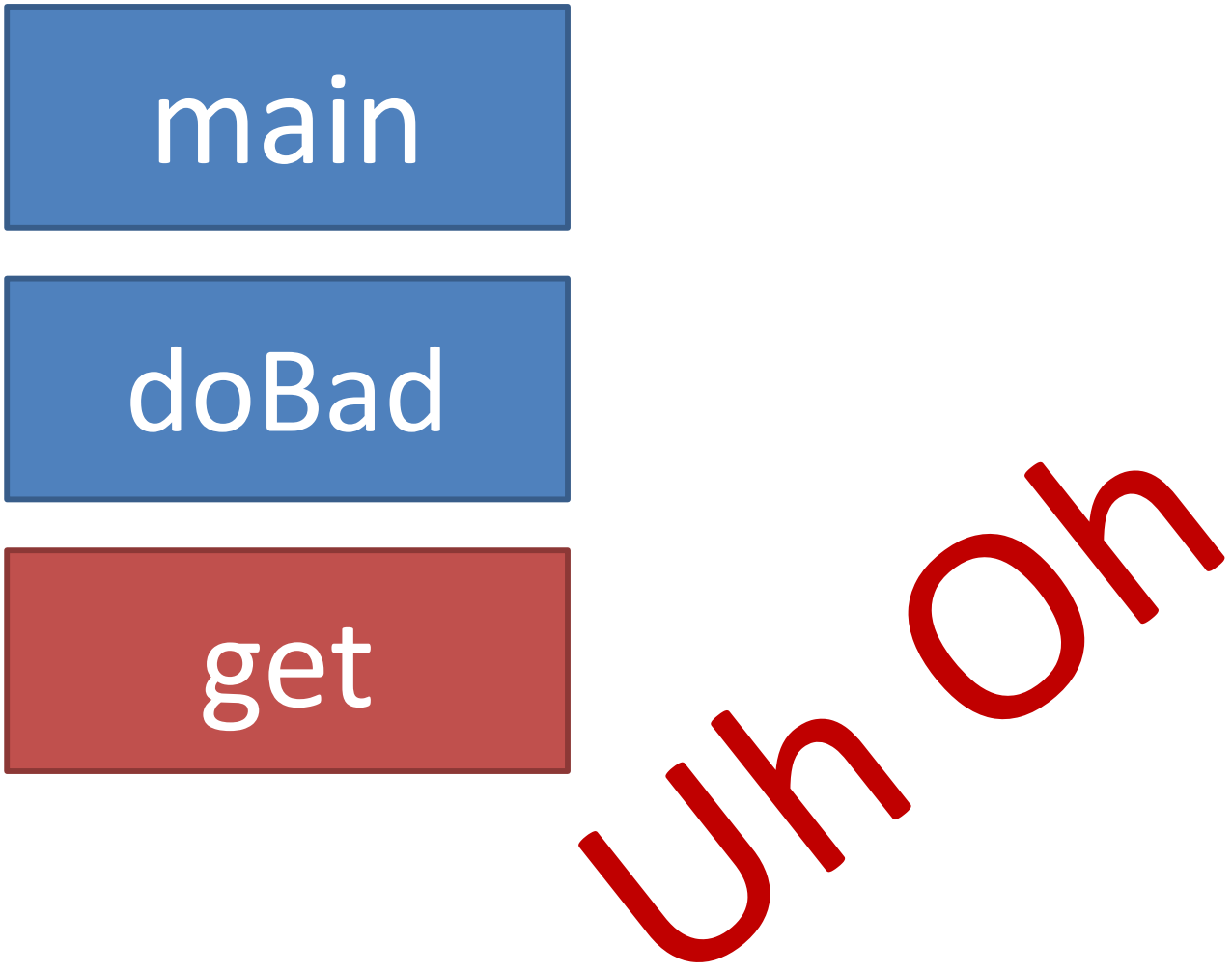
Rethrowing it



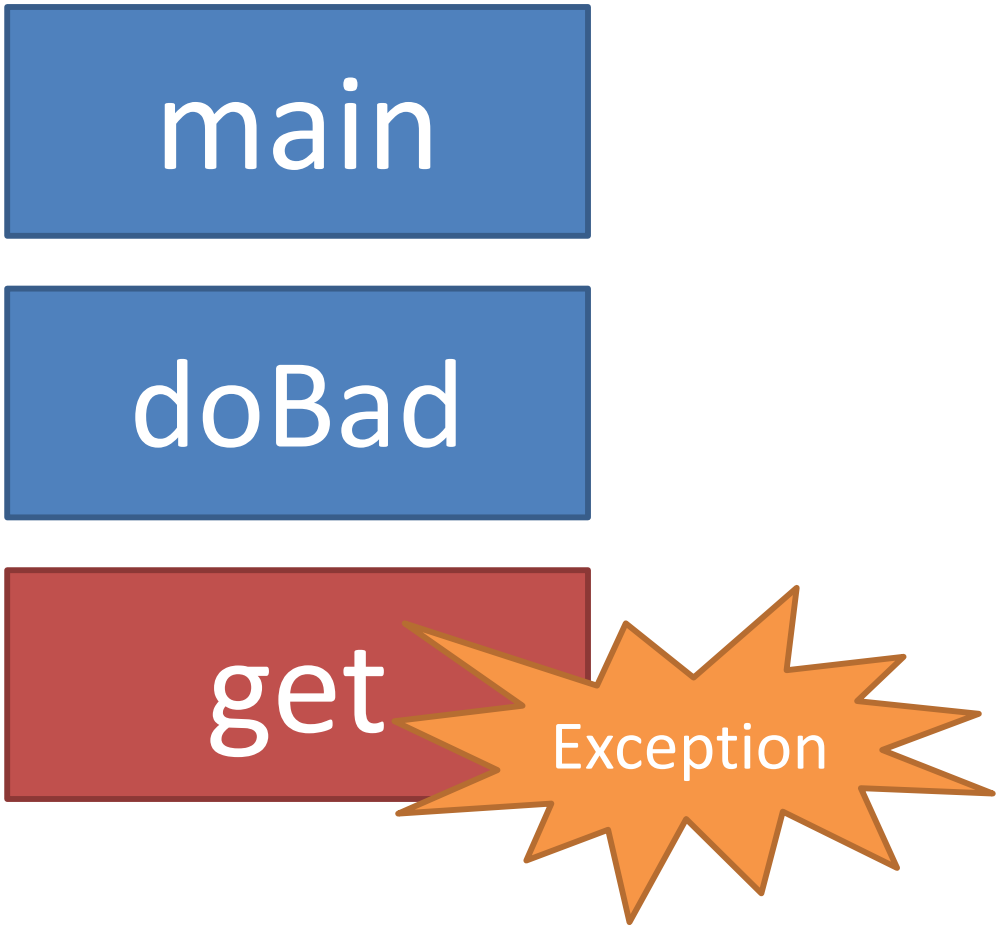
Rethrowing it



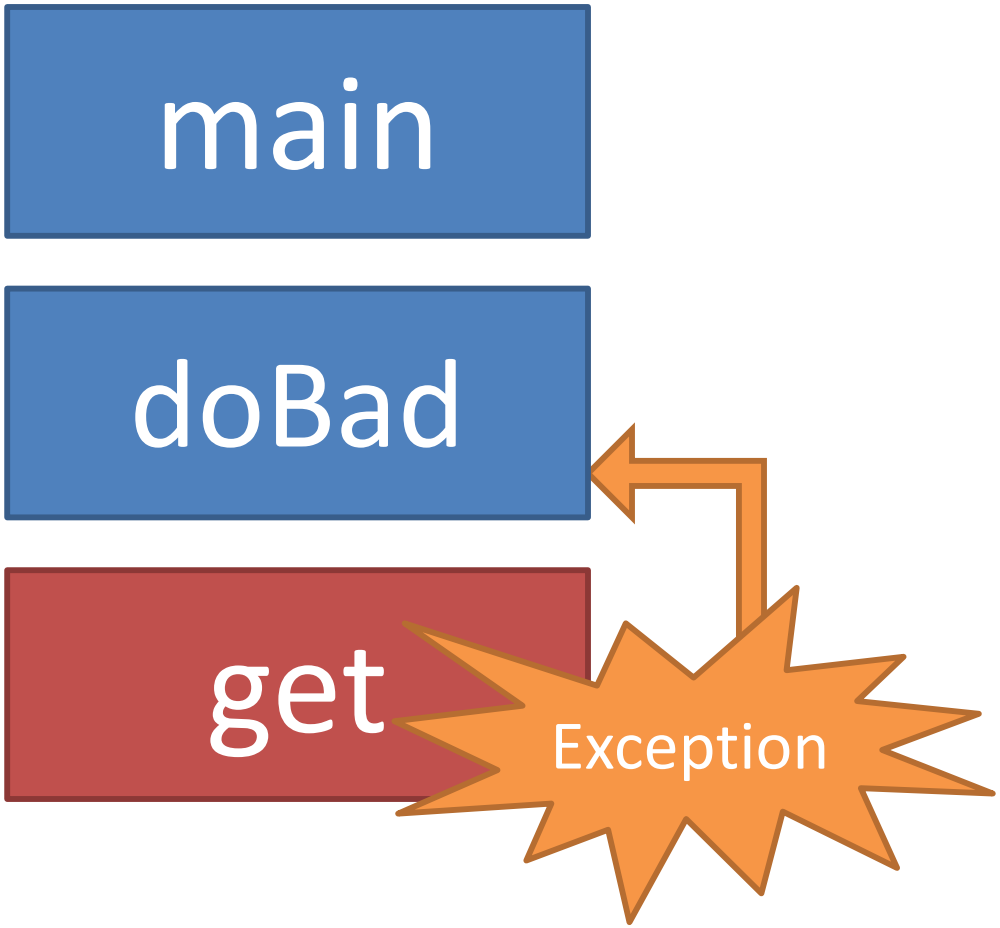
Rethrowing it



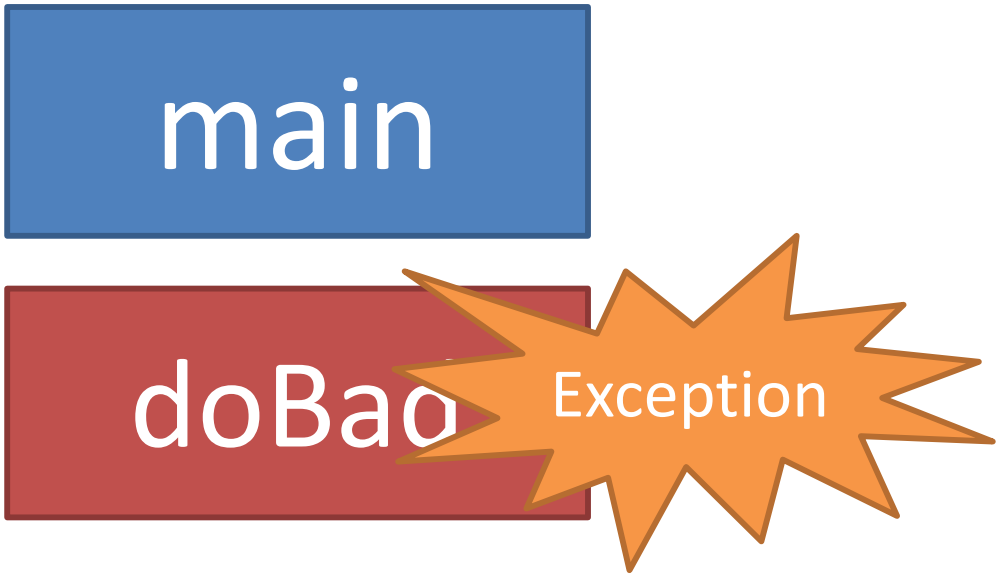
Rethrowing it



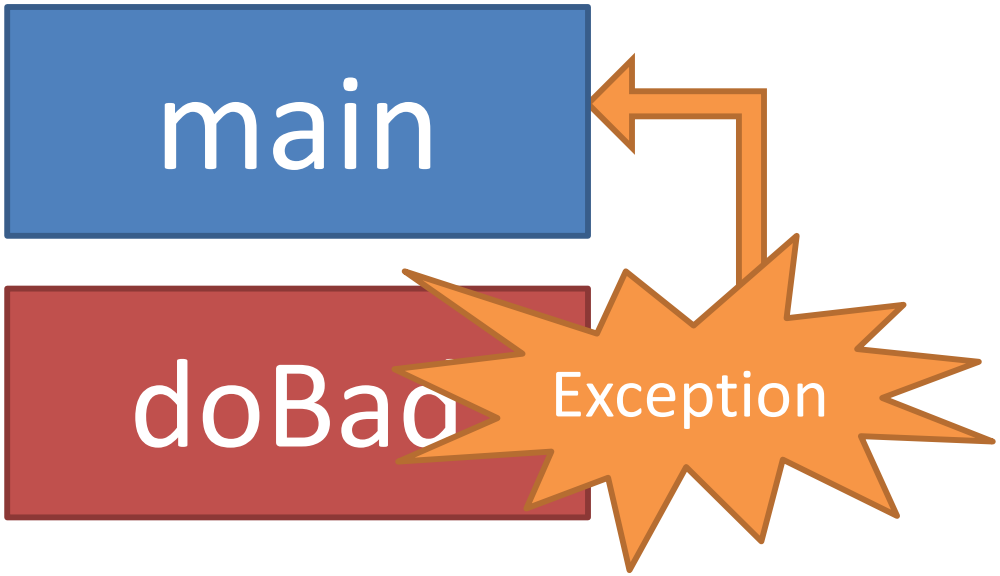
Rethrowing it



Rethrowing it

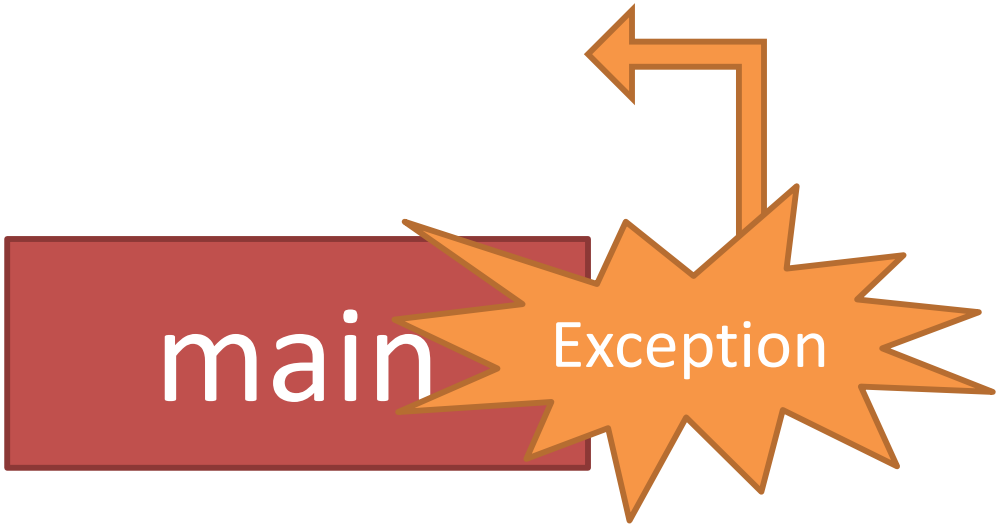


Rethrowing it



Rethrowing it

Rethrowing it



What it no one catches it?

- If you ran

```
public static void main(String[] args) throws Exception {
  doBad();
}
```

- Java will print that error message you see

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: -1
    at YourClass.get(YourClass.java:50)
    at YourClass.doBad(YourClass.java:11)
    at YourClass.main(YourClass.java:10)
```

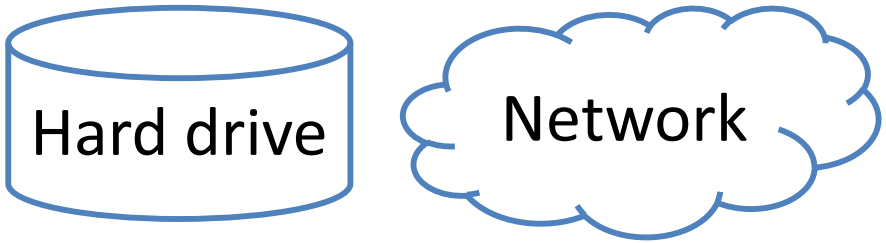
Outline

- Access control. Class scope
- Exceptions
- I/O

We've seen Output

```
System.out.println("some string");
```

The Full Picture



1001010101010001010 …

InputStream

`System.in`

‘O’ ‘k’ ‘a’ ‘y’ ‘ ‘ ‘a’ ‘w’ ‘e’ …

InputStreamReader

“Okay awesome, cool\n” …

BufferedReader

InputStream

- InputStream is a stream of bytes
  - Read one byte after another using `read()`
- A byte is just a number
  - Data on your hard drive is stored in bytes
  - Bytes can be interpreted as characters, numbers..

```
InputStream stream = System.in;
```

InputStreamReader

- Reader is a class for character streams
  - Read one character after another using `read()`
- InputStreamReader takes an InputStream and converts bytes to characters
- Still inconvenient
  - Can only read a character at a time

```
new InputStreamReader(stream)
```

BufferedReader

- BufferedReader buffers a character stream so you can read line by line
  - `String readLine()`

```
new BufferedReader(
    new InputStreamReader(System.in));
```

User Input

```
InputStreamReader ir = new
    InputStreamReader(System.in);
BufferedReader br = new BufferedReader(ir);

br.readLine();
```

FileReader

- FileReader takes a text file
  - converts it into a character stream
  - FileReader(“PATH TO FILE”);

- Use this + BufferedReader to read files!

```
FileReader fr = new FileReader(“readme.txt”);
BufferedReader br = new BufferedReader(fr);
```

FileReader Code

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadFile {

  public static void main(String[] args)  throws IOException{
    // Path names are relative to project directory (Eclipse Quirk )
    FileReader fr = new FileReader("./src/readme");
    BufferedReader br = new BufferedReader(fr);
    String line = null;
    while ((line = br.readLine()) != null) {
      System.out.println(line);
    }
    br.close();

  }
}
```