Access Control, Class scope, Packages and Java APIs

Lets review the static objects again

Overview

- Review
- Access control
- Class scope
- Packages
- Java API

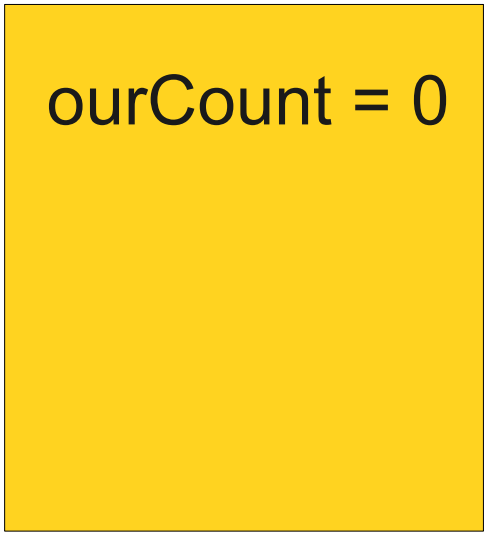
```java
public class Counter {
   int myCount = 0;
   static int ourCount = 0;
   void increment() {
      myCount++;
      ourCount++;
   }
   public static void main(String[] args) {
      Counter counter1 = new Counter();
      Counter counter2 = new Counter();
      counter1.increment();
      counter1.increment();
      counter2.increment();
      System.out.println("Counter 1: " +
counter1.myCount + " " + counter1.ourCount);
      System.out.println("Counter 2: " +
counter2.myCount + " " + counter2.ourCount);
   }
}
```

```java
public class Counter {
   int myCount = 0;
   static int ourCount = 0;
   void increment() {
      myCount++;
      ourCount++;
   }
   public static void main(String[] args) {
      Counter counter1 = new Counter();
      Counter counter2 = new Counter();
      counter1.increment();
      counter1.increment();
      counter2.increment();
      System.out.println("Counter 1: " +
counter1.myCount + " " + counter1.ourCount);
      System.out.println("Counter 2: " +
counter2.myCount + " " + counter2.ourCount);
   }
}
```

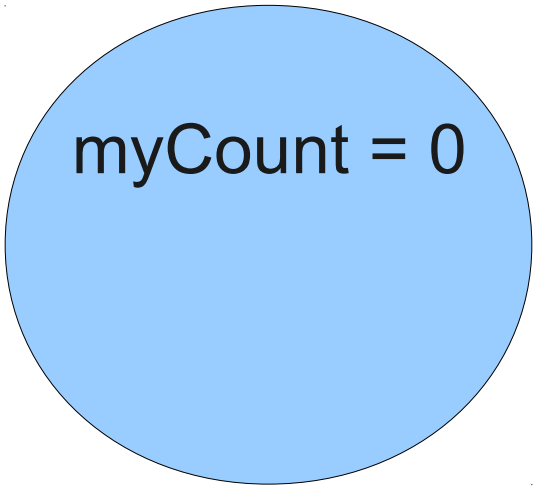
Fields

Method

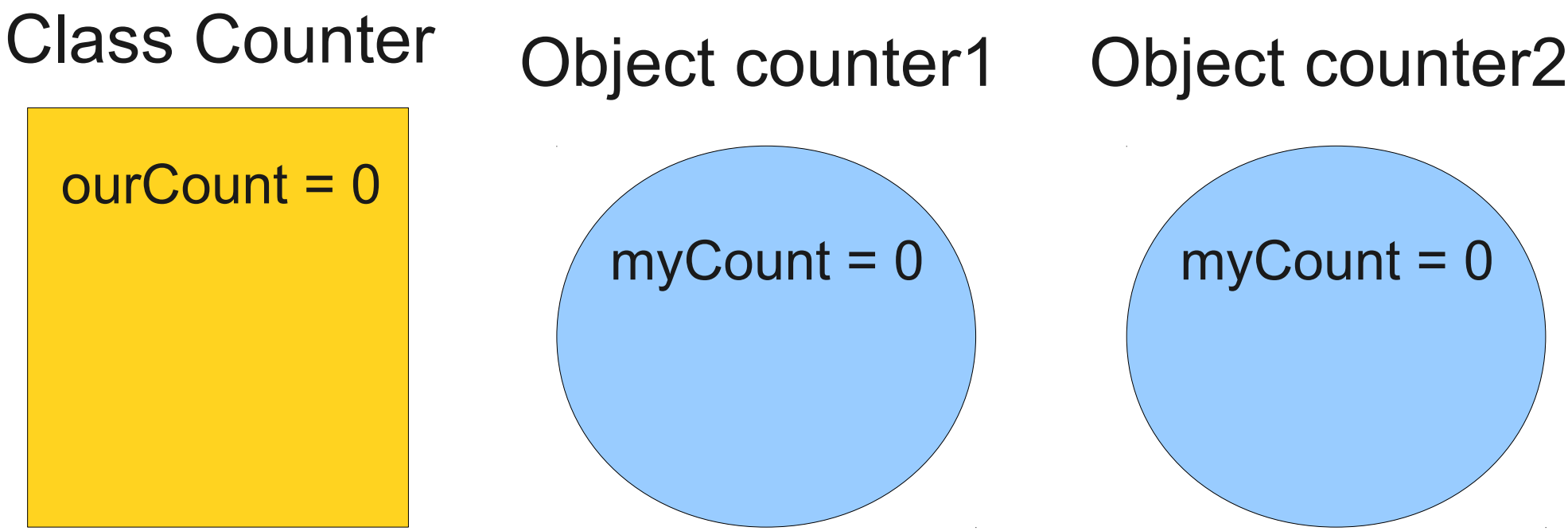
Class Counter



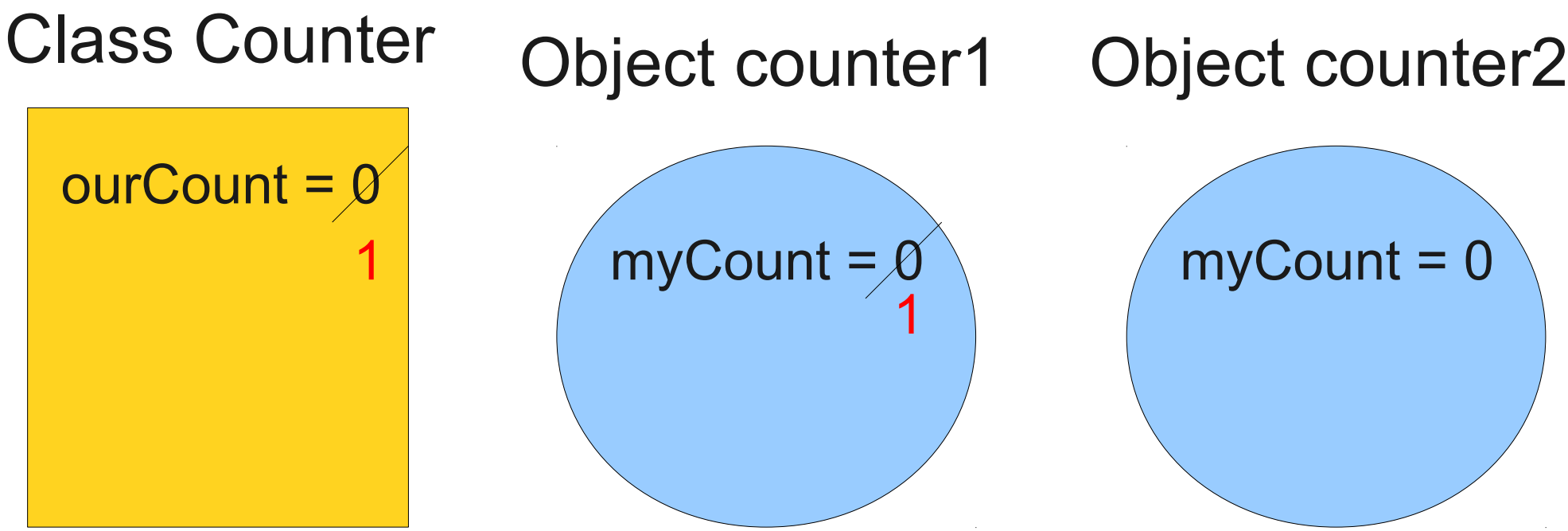
Object counter1



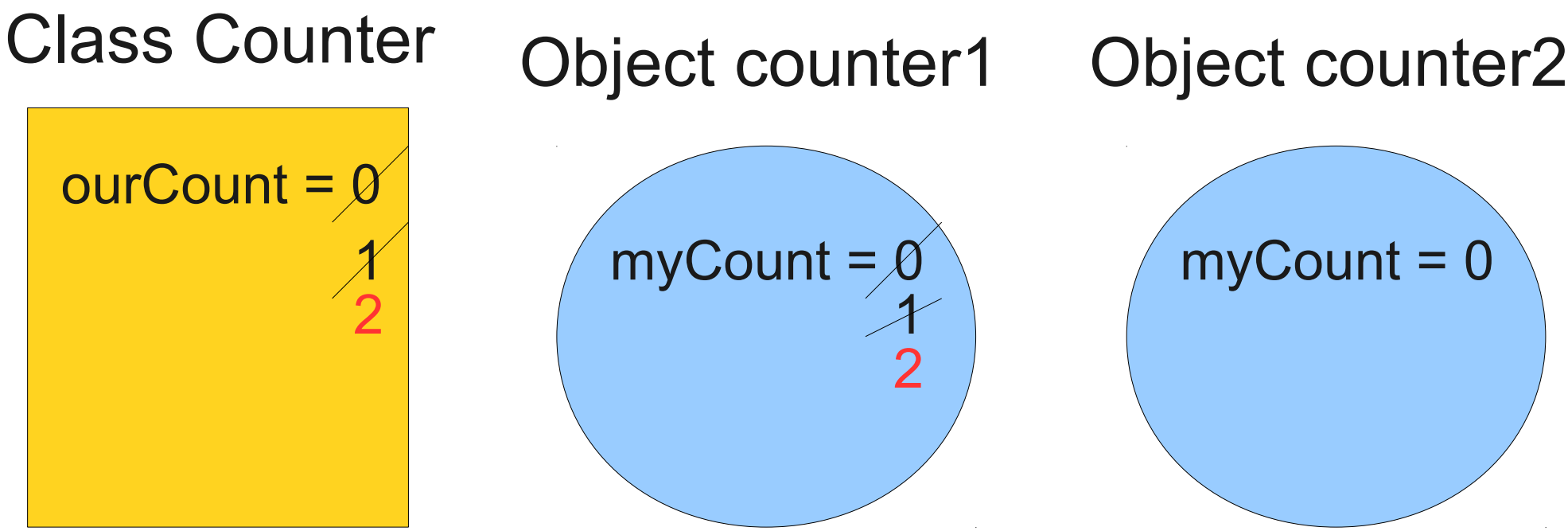

```
Counter counter1 = new Counter();
```



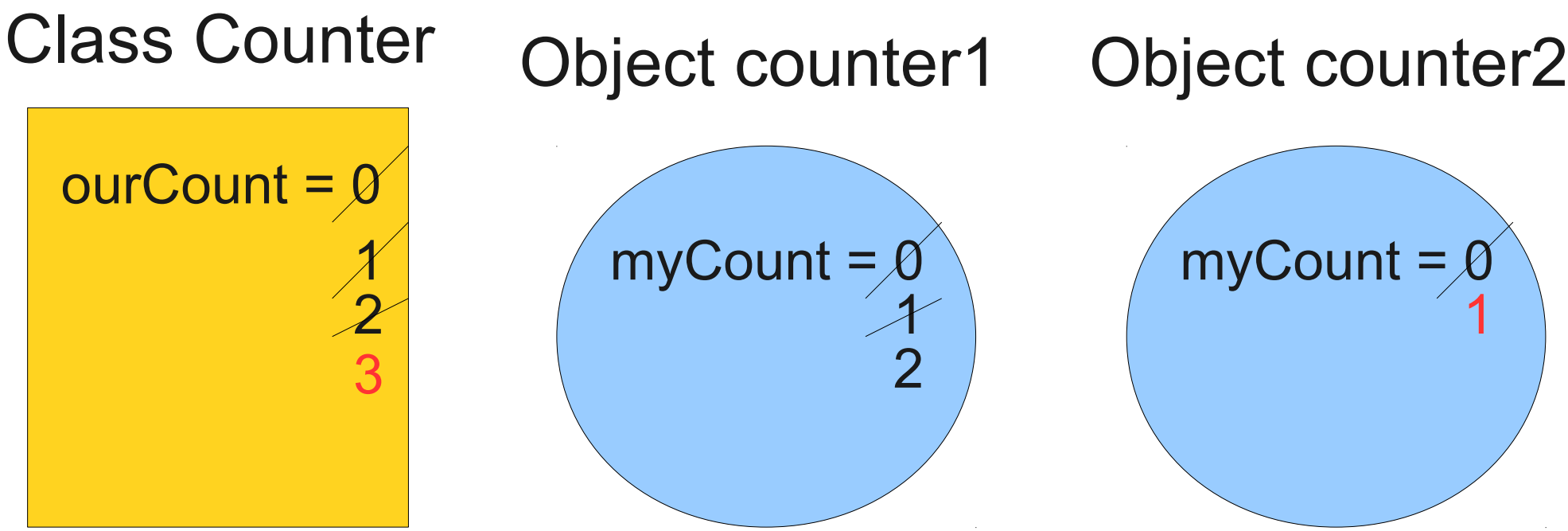

```
Counter counter1 = new Counter();
Counter counter2 = new Counter();
```




```
Counter counter1 = new Counter();
Counter counter2 = new Counter();
counter1.increment();
```




```
Counter counter1 = new Counter();
Counter counter2 = new Counter();
counter1.increment();
counter1.increment();
```




```
Counter counter1 = new Counter();
Counter counter2 = new Counter();
counter1.increment();
counter1.increment();
counter2.increment();
```

Why Access Control

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

Overview

- Review
- Access control
- Class scope
- Packages
- Java API

Scope Review

```
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);
   }
}
```

Scope Review

```
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);
  }
}
```

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);
  }
}
```

Scope

Just like methods, variables are accessible inside {}

- Previous lessons: method-level scope

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

- This lesson: class-level scope

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

'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;
  }
}
```

Overview

- Review
- Access control
- Class scope
- Packages
- Java API

Packages

- Each class belongs to a package
- Classes in the same package serve a similar purpose
- Packages are just directories
- Classes in other packages need to be imported

Defining Packages

```
package path.to.package.foo;
class Foo {
    ...
}
```

Using Packages

```
import path.to.package.foo.Foo;
import path.to.package.foo.*;
```

```
package parenttools;

public class BabyFood {

}
```

```
package parenttools;

public class Baby {

}
```

```
package adult;

import parenttools.Baby;
import parenttools.BabyFood;

public class Parent {
 public static void main(String[] args) {
    Baby baby = new Baby();
    baby.feed(new BabyFood());
 }
}
```

Why Packages?

- Combine similar functionality
  - org.boston.libraries.Library
  - org.boston.libraries.Book
- Separate similar names
  - shopping.List
  - packing.List

Special Packages

All classes “see” classes in the same package (no import needed)

All classes “see” classes in java.lang

Example: java.lang.String; java.lang.System

Overview

- Review
- Access control
- Class scope
- Packages
- Java API

Java API

Java includes lots of packages/classes

Reuse classes to avoid extra work

http://java.sun.com/javase/6/docs/api/

Arrays with items

Create the array bigger than you need

Track the next “available” slot

```
Book[] books = new Book[10];
int nextIndex = 0;

books[nextIndex] = b;
nextIndex = nextIndex + 1;
```

Arrays with items

Create the array bigger than you need

Track the next “available” slot

```
Book[] books = new Book[10];
int nextIndex = 0;

books[nextIndex] = b;
nextIndex = nextIndex + 1;
```

What if the library expands?

ArrayList

Modifiable list

Internally implemented with arrays

Features

- Get/put items by index
- Add items
- Delete items
- Loop over all items

Array → ArrayList

```
Book[] books =
     new Book[10];
int nextIndex = 0;

books[nextIndex] = b;
nextIndex += 1;
```

→

```
ArrayList<Book> books
= new ArrayList<Book>();

books.add(b);
```

```java
import java.util.ArrayList;
class ArrayListExample {
    public static void main(String[] arguments) {
        ArrayList<String> strings = new ArrayList<String>();
        strings.add("Evan");
        strings.add("Eugene");
        strings.add("Adam");

        System.out.println(strings.size());
        System.out.println(strings.get(0));
        System.out.println(strings.get(1));

        strings.set(0, "Goodbye");
        strings.remove(1);



        for (String s : strings) {
            System.out.println(s);
        }
    }
}
```

Sets

Like an ArrayList, but

- Only one copy of each object, and
- No array index

Features

- Add objects to the set
- Remove objects from the set
- Is an object in the set?

TreeSet: Sorted (lowest to highest)

HashSet: Unordered (pseudo-random)

```java
import java.util.TreeSet;

class SetExample {
    public static void main(String[] arguments) {
        TreeSet<String> strings = new TreeSet<String>();
        strings.add("Evan");
        strings.add("Eugene");
        strings.add("Adam");

        System.out.println(strings.size());
        System.out.println(strings.first());
        System.out.println(strings.last());

        strings.remove("Eugene");

        for (String s : strings) {
            System.out.println(s);
        }
    }
}
```

Maps

Stores a (*key*, *value*) pair of objects

Look up the *key*, get back the *value*

Example: Address Book

- Map from names to email addresses

TreeMap: Sorted (lowest to highest)

HashMap: Unordered (pseudo-random)

```java
public static void main(String[] arguments) {
    HashMap<String, String> strings = new HashMap<String, String>();
    strings.put("Evan", "email1@mit.edu");
    strings.put("Eugene", "email2@mit.edu");
    strings.put("Adam", "email3@mit.edu”);

    System.out.println(strings.size());
    strings.remove("Evan");
    System.out.println(strings.get("Eugene"));

    for (String s : strings.keySet()) {
        System.out.println(s);
    }
    for (String s : strings.values()) {
        System.out.println(s);
    }
    for (Map.Entry<String, String> pairs : strings.entrySet()) {
        System.out.println(pairs);
    }
}
```

Warnings

Using TreeSet/TreeMap?

Read about Comparable interface

Using HashSet/HashMap?

Read about equals, hashCode methods

Note: This only matters for classes you build, not for java built-in types.

Summary

- Review
- Access control
- Class scope
- Packages
- Java API

An Oval Sprite

```
public class Oval implements Sprite {
    private int width, height;
    private Color color;

    public Oval(int width, int height, Color color) {
        // set the fields ...
    }

    public void draw(Graphics surface, int x, int y) {
        surface.setColor(color);
        surface.fillOval(x, y, width, height);
        surface.drawOval(x, y, width, height);
    }
   ...
}
```

A Mover that doesn’t bounce

```
public class StraightMover {
    private int x, y, xDirection, yDirection;
    private Sprite sprite;

    public StraightMover(int startX, int startY, Sprite sprite) {
        x = startX;
        y = startY;
        this.sprite = sprite;
    }
    public void setMovementVector(int xIncrement, int yIncrement) {
        xDirection = xIncrement;
        yDirection = yIncrement;
    }
    public void draw(Graphics graphics) {
        sprite.draw(graphics, x, y);
        x += xDirection;
        y += yDirection;
    }
}
```

Inheritance

Exceptions

I/O

Inheritance

Very *Very* Basic Inheritance

- Making a Game

```
public class Dude {
    public String name;
    public int hp = 100
    public int mp = 0;

    public void sayName() {
          System.out.println(name);
    }
    public void punchFace(Dude target) {
          target.hp -= 10;
    }
}
```

Inheritance..

- Now create a Wizard…

```
public class Wizard {
    // ugh, gotta copy and paste
    // Dude's stuff
}
```

Inheritance?

- Now create a Wizard…

But Wait!

A Wizard does and has everything a Dude does and has!

Inheritance?

- Now create a Wizard…

Don’t Act Now!

You don’t have to Copy & Paste!

Buy Inheritance!

- Wizard is a **subclass** of Dude

```
public class Wizard extends Dude {
}
```

Buy Inheritance!

- Wizard can use everything* the Dude has!

```
wizard1.hp += 1;
```

- Wizard can do everything* Dude can do!

```
wizard1.punchFace(dude1);
```

- You can use a Wizard like a Dude too!

```
dude1.punchface(wizard1);
```

*except for private fields and methods

Buy Inheritance!

- Now augment a Wizard

```
public class Wizard extends Dude {
    ArrayList<Spell> spells;
    public class cast(String spell) {
        // cool stuff here
        ...
        mp -= 10;
    }
}
```

Inheriting from inherited classes

- What about a Grand Wizard?

```
public class GrandWizard extends Wizard {
    public void sayName() {
      System.out.println("Grand wizard" + name)
    }
}

grandWizard1.name = "Flash"
grandWizard1.sayName();
((Dude)grandWizard1).sayName();
```

How does Java do that?

- What Java does when it sees

```
grandWizard1.punchFace(dude1)
```

1. Look for `punchFace()` in the GrandWizard class
2. It's not there! Does GrandWizard have a parent?
3. Look for `punchFace()` in Wizard class
4. It's not there! Does Wizard have a parent?
5. Look for `punchFace()` in Dude class
6. Found it! Call `punchFace()`
7. Deduct hp from dude1

How does Java do that? pt2

- What Java does when it sees

```
((Dude)grandWizard1).sayName()
```

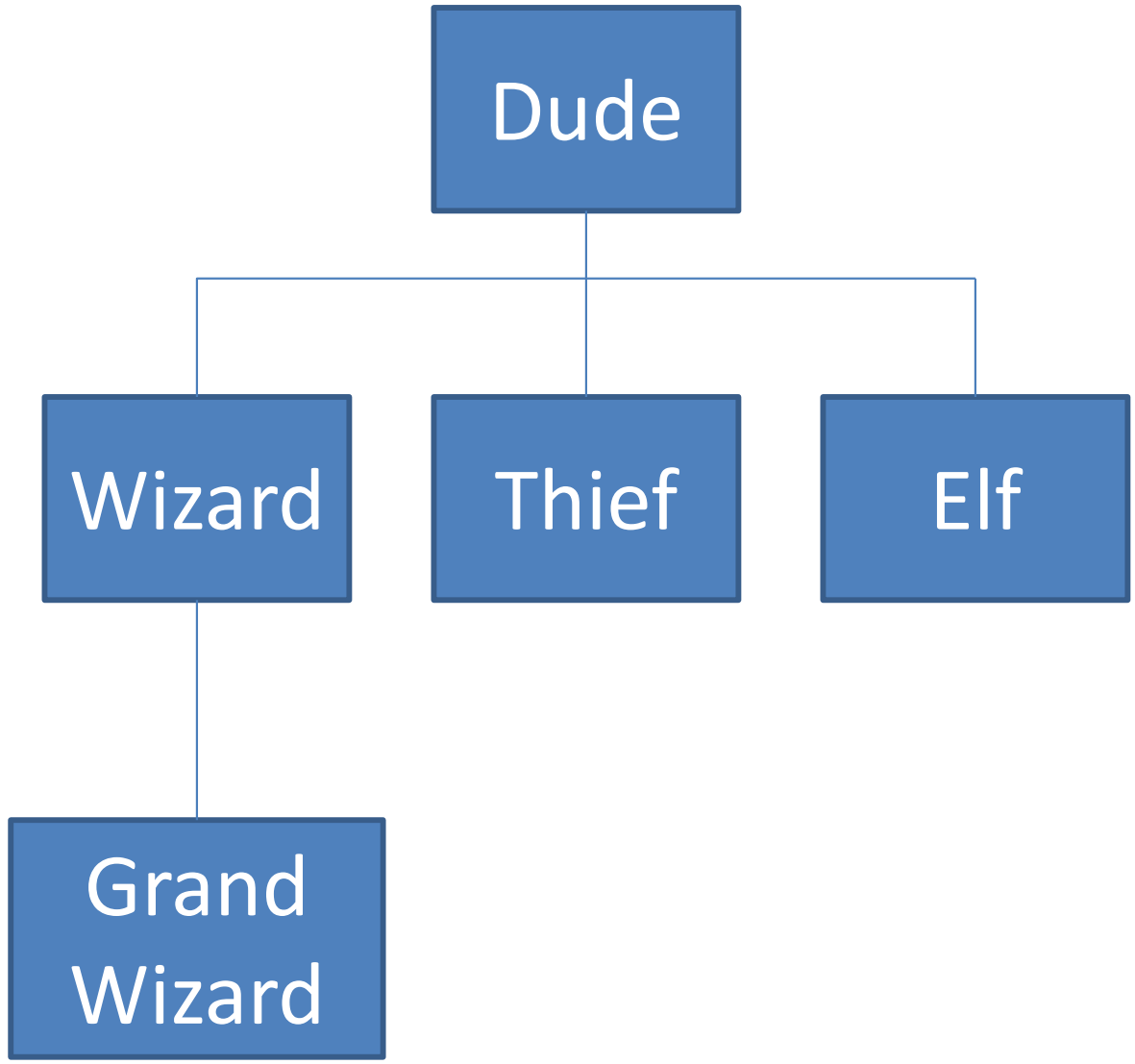
1. Cast to `Dude` tells Java to start looking in `Dude`
2. Look for `sayName()` in Dude class
3. Found it! Call `sayName()`

What's going on?

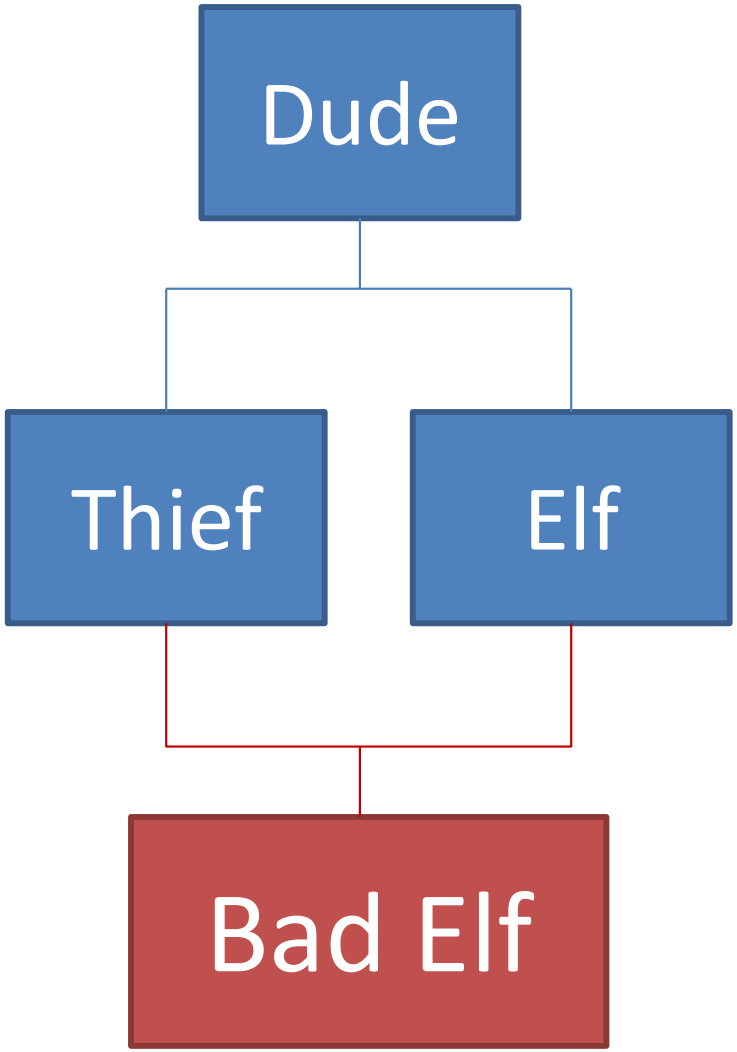
Parent of Wizard, Elf..

Subclass of Dude

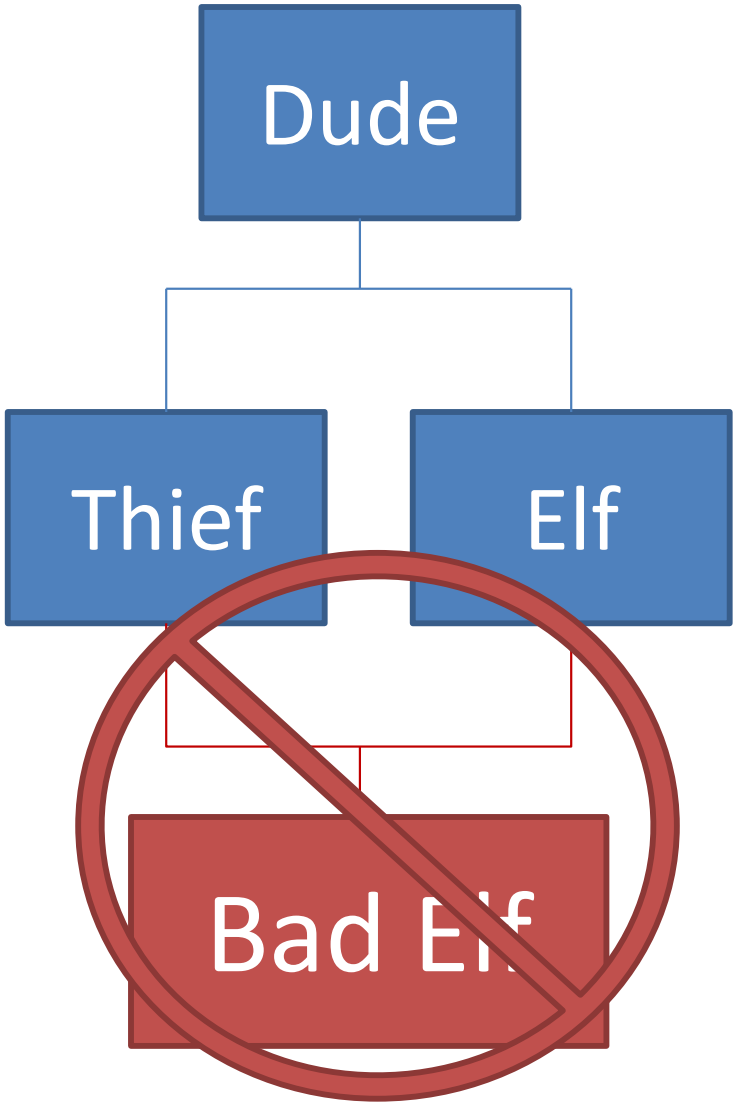
Subclass of Wizard



You can only inherit from one class



You can only inherit from one class



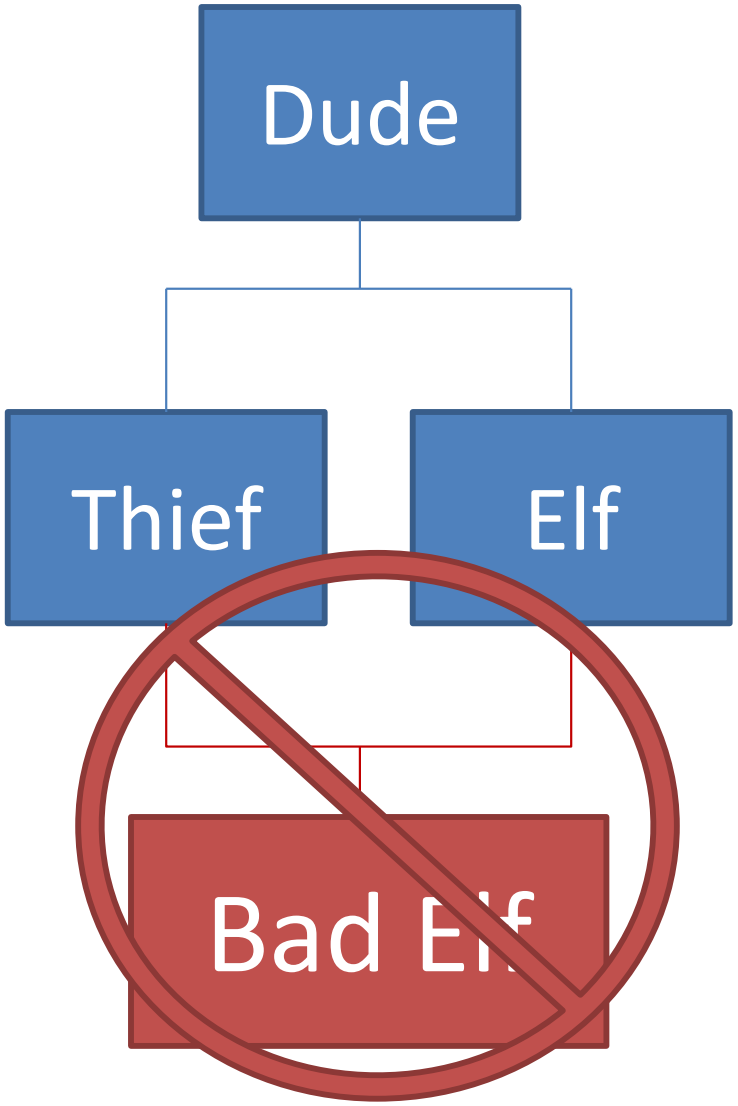
You can only inherit from one class

What if Thief and Elf both implement

```
public void sneakUp()
```

If they implemented differently, which `sneakUp()` does BadElf call?

Java Doesn't Know!!



Inheritance Summary

- class A extends B {} == A is a subclass of B
- A has all the fields and methods that B has
- A can add it's own fields and methods
- A can only have 1 parent
- A can replace a parent's method by re-implementing it
- If A doesn't implement something Java searches ancestors

So much more to learn!

- http://java.sun.com/docs/books/tutorial/java/IandI/subclasses.html
- http://home.cogeco.ca/~ve3ll/jatutor5.htm
- http://en.wikipedia.org/wiki/Inheritance_(computer_science)
- http://www.google.com

Exceptions

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!");
}
```

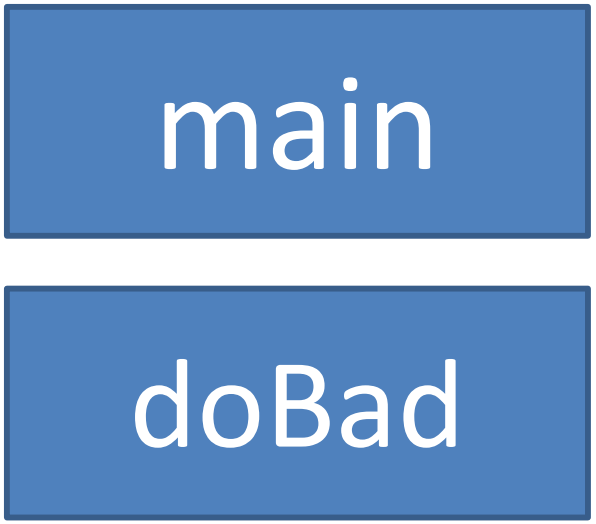
Rethrowing it

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

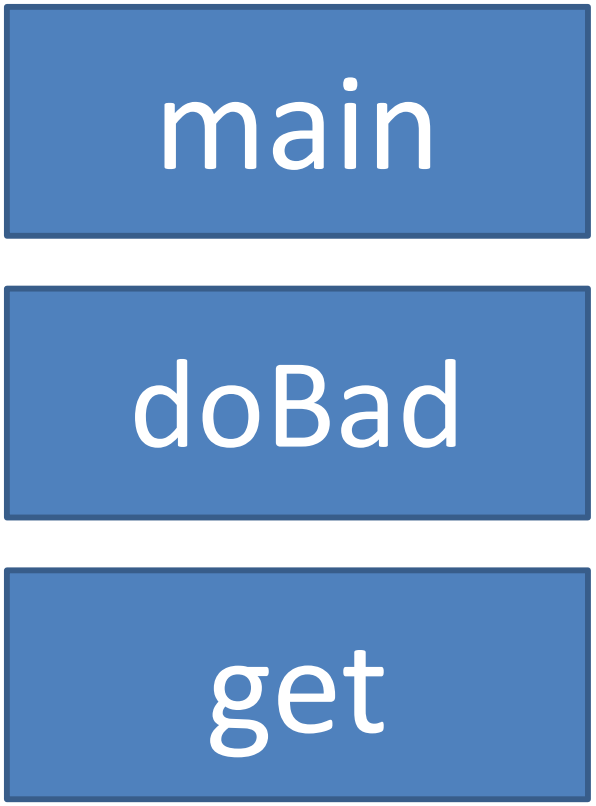
```
void doBad() throws ArrayOutOfBoundsException {
    get(-1);
}
```

Rethrowing it

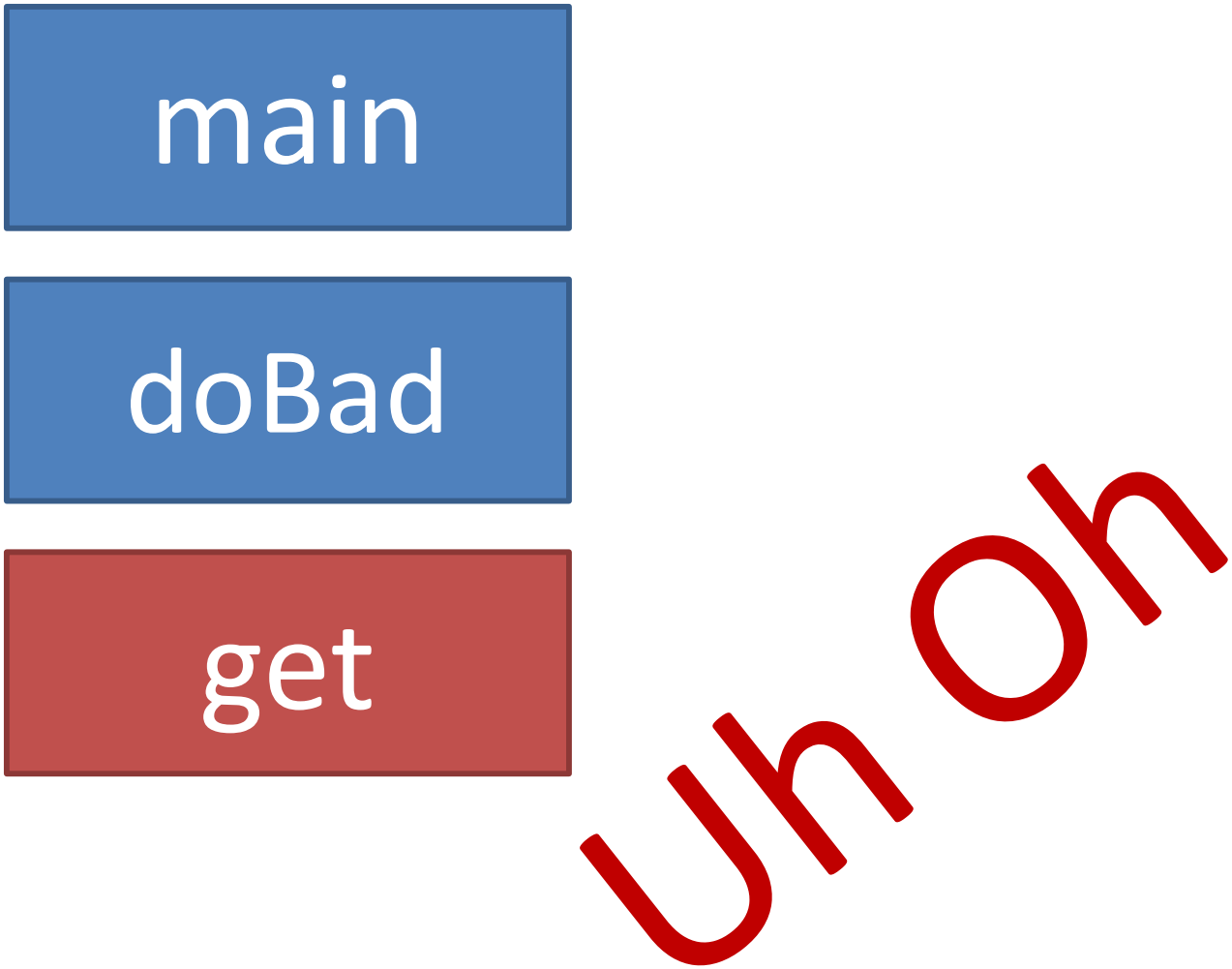
Rethrowing it



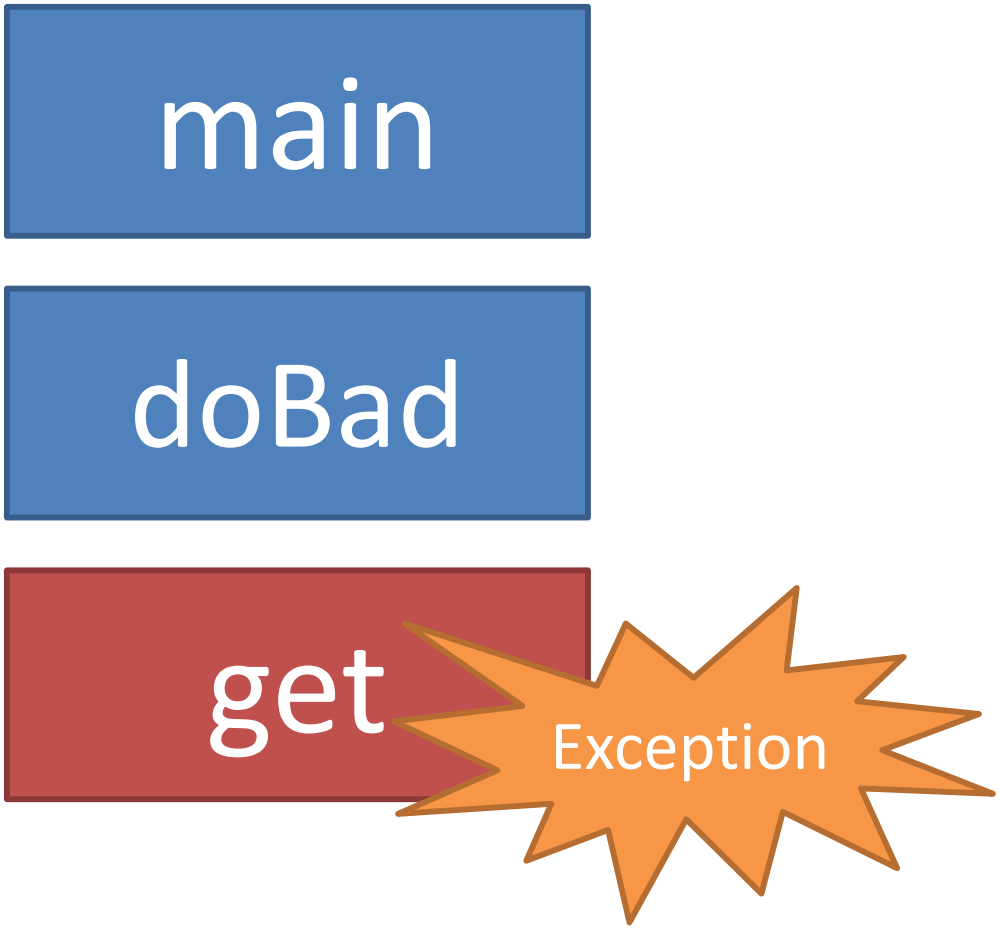
Rethrowing it



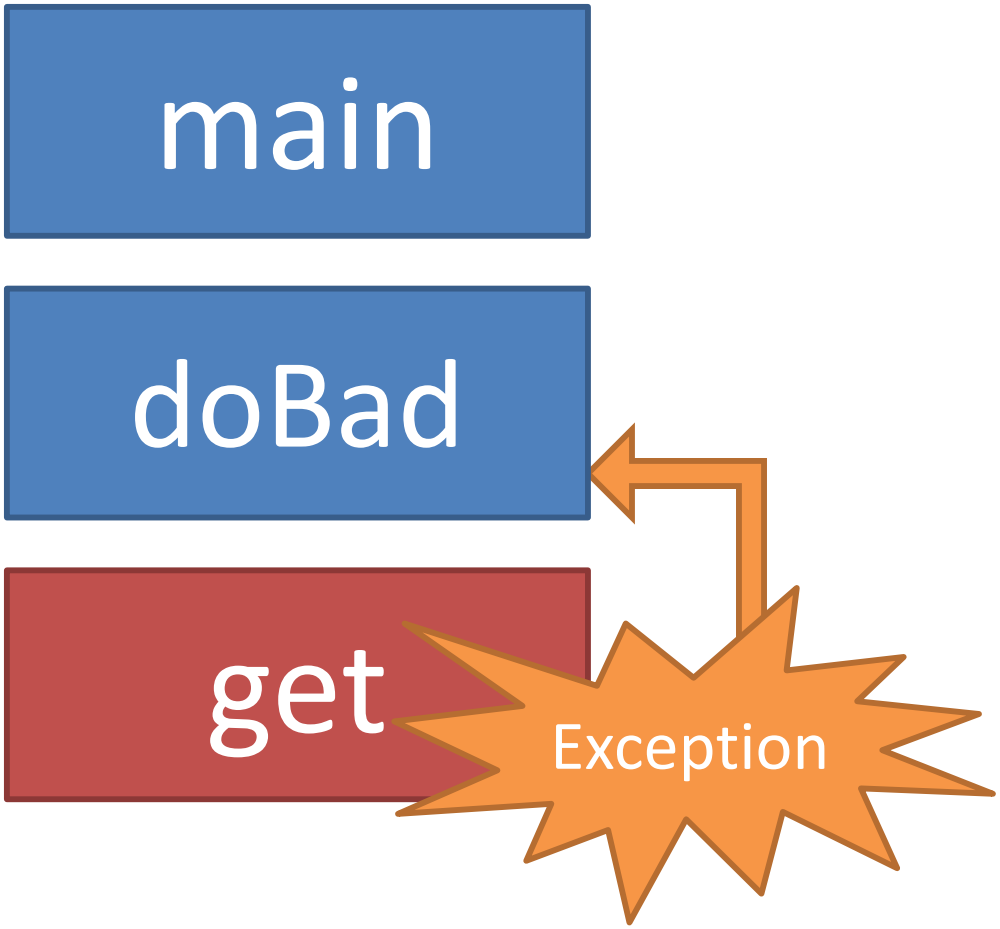
Rethrowing it



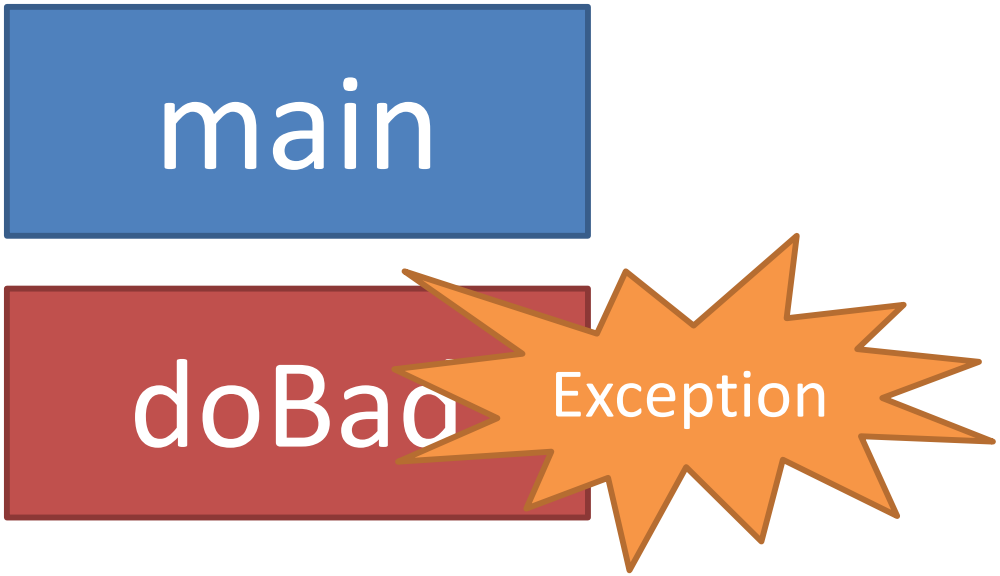
Rethrowing it



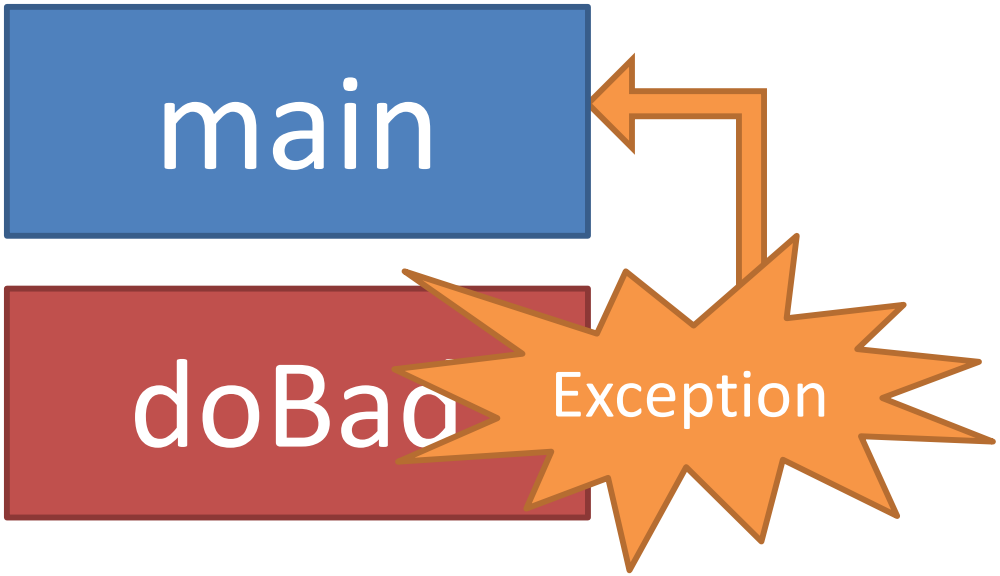
Rethrowing it



Rethrowing it

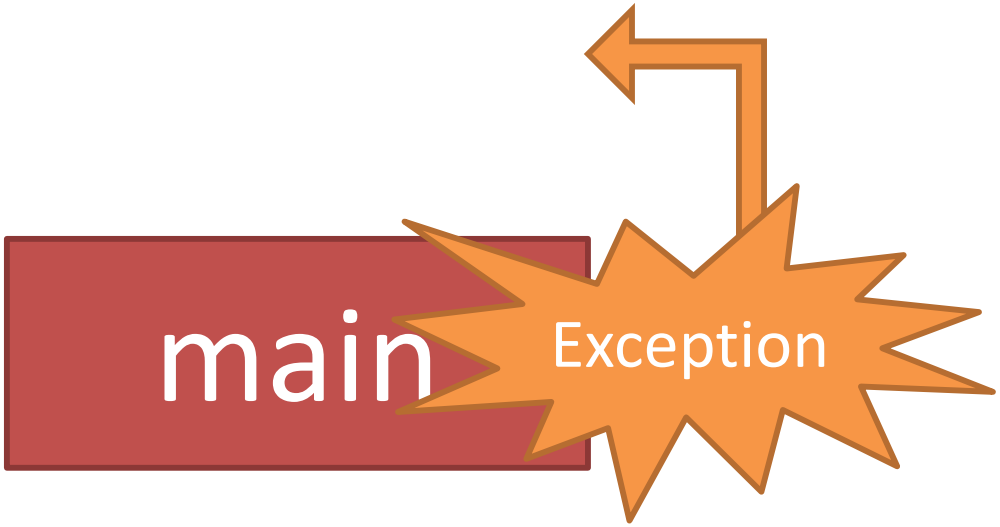


Rethrowing it



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)
```

More Info?

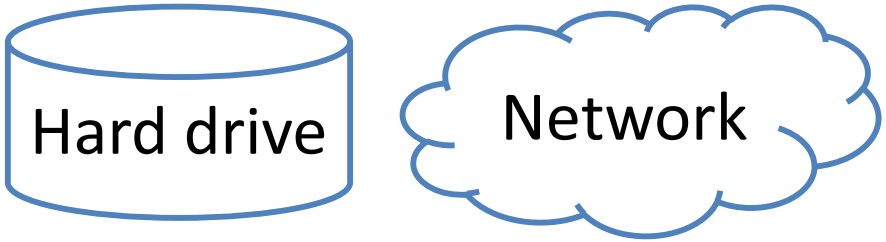
- http://java.sun.com/docs/books/tutorial/essential/exceptions
- http://en.wikipedia.org/wiki/Exceptions

I/O

We’ve seen Output

```
System.out.println(“some string”);
```

The Full Picture



10010101010101000101 …

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();

  }
}
```

More about I/O

- http://java.sun.com/docs/books/tutorial/essential/io/