Week 3

Exceptions, Access modifiers, Weird classes

Project 1 is out!

Wildlife of the Plain

- Due Saturday, February 9th
- Focuses on object orientation principles
  - Review last weeks slides!!!
- Must write test cases

```
F5 E  E  F0 E  E
B3 F1 B0 R0 G  R0
R0 E  R2 B0 B2 G
B0 E  E  R1 F0 E
B1 E  E  G  E  R0
G  G  E  B0 R2 E
```

How to zip your homeworks to turn them in

1. In Eclipse, select File -> Export, General -> Archive File, hit Next
2. In left hand menu, select only your homework project
3. In “To archive file”, select temporary location to put the zip file. Name the zip file “Firstname_Lastname_hw1.zip”
4. In options, select “Save in zip format”, and “Create directory structure for files”, then hit Finish
5. Open up your newly created .zip file, and **MAKE SURE THAT ALL YOUR .JAVA FILES ARE THERE!** If you turn in an unopenable .zip file, or a .zip file without your .java files, **YOU WILL GET A ZERO!**
6. Turn in homework!

Exam 1 is February 20th!

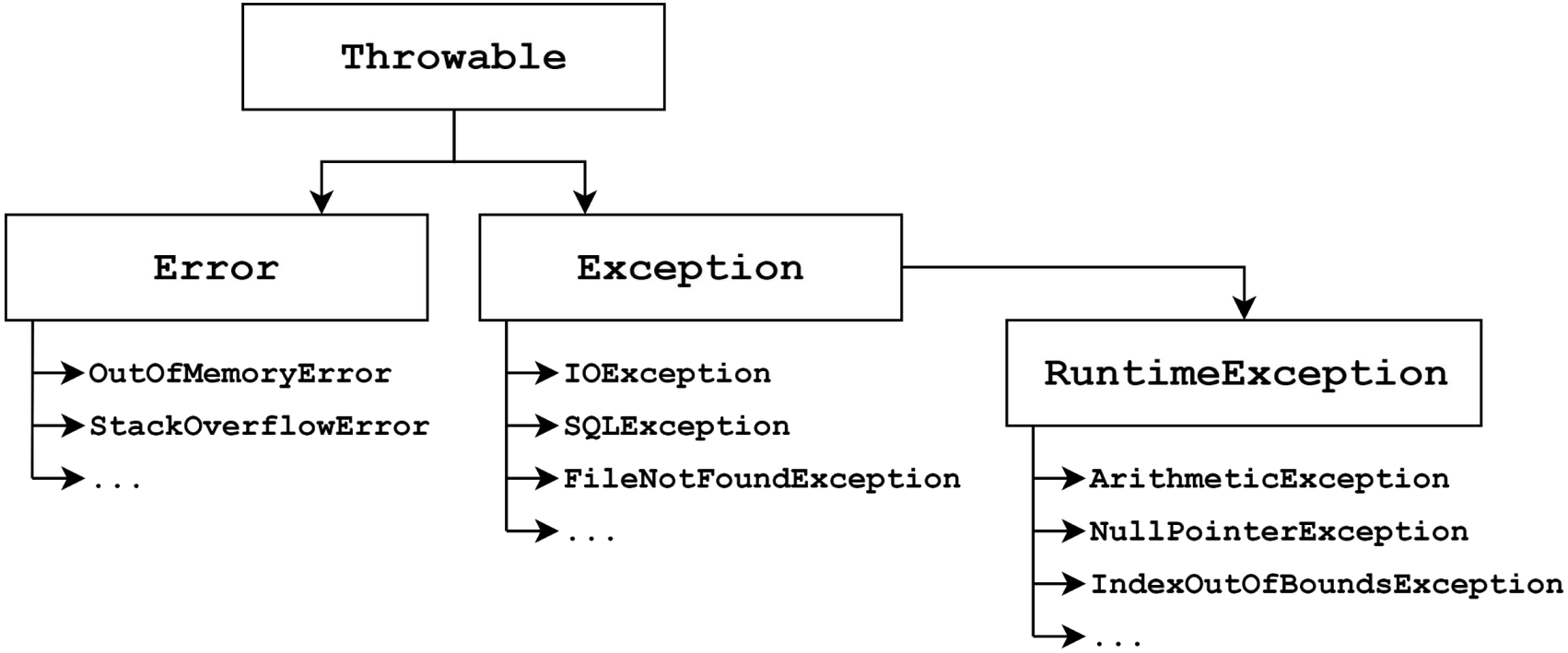
Beware.

And come to recitation for exam review!

Exceptions

What are exceptions?

Simply, an error during runtime



Source: http://www.techburps.com/core-java-tutorials/exception-handling/115

Runtime Exceptions

- Regular exceptions thrown during runtime
- Can be **caught**, but doesn't need to be most times
- **Unchecked**

Example: ArithmeticException

```
int a = 7;
int b = a/0;
```

Example: NullPointerException

```
List<Integer> l = null;
l.add(1);
```

Exceptions

- Other exceptions thrown during runtime
- **Checked** during compile-time, meaning they must be explicitly handled (either caught or thrown)
- Java's way of saying "It's important that this exception be handled if thrown"

Example: FileNotFoundException

```
public static int getNumber(String filename)
    throws FileNotFoundException {

    File f = new File(filename);
    Scanner s = new Scanner(f);

    int n = s.nextInt();

    s.close();

    return n;
}
```

Errors

- “An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to **catch**.”
- **Unchecked**

Example: OutOfMemoryError

```
public static void main(String[] args)
{
    List<Integer> list = new ArrayList<>();

    while(true) {
        list.add(1);
    }
}
```

Source: https://docs.oracle.com/javase/8/docs/api/java/lang/Error.html

Try/Catch/Finally

Try: Code which may throw an exception

Catch: Executes if an exception in the try block is thrown

Finally: Executes regardless of if an exception is thrown or not

```
try {

} catch(Exception e) {

} finally {

}
```

Try/Catch/Finally

Code:

```
try {
    int a = 8/0;
} catch(ArithmeticException e) {
    System.err.println("Caught ArithmeticException: "
            + e.getMessage());
} finally {
    System.out.println("Finished with math!");
}
```

Output:

Try/Catch/Finally

Code:

```
try {
    int a = 8/0;
} catch(ArithmeticException e) {
    System.err.println("Caught ArithmeticException: "
            + e.getMessage());
} finally {
    System.out.println("Finished with math!");
}
```

Output:

```
Caught ArithmeticException: / by zero
Finished with math!
```

Try/Catch/Finally: Checked Exceptions

Before:

```java
public static int getNumber(String filename)
        throws FileNotFoundException {

    File f = new File(filename);
    Scanner s = new Scanner(f);

    int n = s.nextInt();

    s.close();

    return n;
}
```

After:

```java
public static int getNumber(String filename) {

    File f = new File(filename);
    Scanner s = null;
    int number = 8; //Default answer

    try {
        s = new Scanner(f); //throws FileNotFound
        number = s.nextInt(); //throws InputMismatch

    } catch(FileNotFoundException e) {
        System.err.println("Caught FileNotFoundException: "
                + e.getMessage());

    } finally {
        if(s != null)
            s.close();
    }

    return number;
}
```

Try/Catch/Finally: Questions

Q: Is the following code legal?

```
try {

} finally {

}
```

Source: https://docs.oracle.com/javase/tutorial/essential/exceptions/QandE/questions.html

Try/Catch/Finally: Questions

Q: Is the following code legal?

A: Yes, and can be useful. Code in the finally block is executed no matter what is thrown in “try”

```
try {

} finally {

}
```

Source: https://docs.oracle.com/javase/tutorial/essential/exceptions/QandE/questions.html

Try/Catch/Finally: Questions

Q: Is there anything wrong with this exception handler? Will this code compile?

```
try {

} catch(Exception e) {

} catch(ArithmeticException a) {

}
```

Source: https://docs.oracle.com/javase/tutorial/essential/exceptions/QandE/questions.html

Try/Catch/Finally: Questions

Q: Is there anything wrong with this exception handler? Will this code compile?

A: The first catch block will catch any exception, including ArithmeticException, the second catch block will never be reached.

This does not compile.

```
try {

} catch(Exception e) {

} catch(ArithmeticException a) {

}
```

Source: https://docs.oracle.com/javase/tutorial/essential/exceptions/QandE/questions.html

Can I throw my own exceptions?

You sure can! And sometimes, you should!

```
public double circleArea(double radius) {

    if(radius < 0) {
        String msg = "Radius must be positive";
        throw new IllegalArgumentException(msg);
    }

    return Math.PI * radius * radius;
}
```

Can I make my own exceptions?

Again, yes! But do this very sparingly...

```
public class MyException extends Exception {

}
```

Access Modifiers

What is an access modifier?

Limits what fields are visible to other code. Can be used for classes, constructors, class fields, methods, interfaces, abstract classes, etc.

```
public int a;
int b;
protected int c;
private int d;
```

	private	default	protected	public
Inside Class				
Class in the same package				
Sub-class in the same package				
Class in some other package				
Sub-class in some other package				

Rules of thumb

- **Public**: For methods which are necessary to effectively use an object, and to almost never be used for fields
- **Protected:** Only if you know a subclass will need it
- **Private:** Use for all your fields, and for methods which are not needed for those using an object (like helper methods)

Source: https://www.javaworld.com/article/2077443/core-java/access-modifiers-what-gives.html

Weird classes!

Inner classes

- Classes defined within another class
- Useful if inner class will <u>only</u> be used by outer class
- Can be private to just the outer class

```
class Outer {

    class Inner {

        //Inner fields and methods...

    }

    //Outer fields and methods...
}
```

Inner classes: Questions

Q: Can an outer class access an inner class' private fields? What about the other way around?

```
class Outer {

    private int a;

    private class Inner {

        private int b;

    }
}
```

Inner classes: Questions

Q: Can an outer class access an inner class' private fields? What about the other way around?

A: Yes, Inner can access Outer private fields, and vice versa.

```
class Outer {

    private int a;

    private class Inner {

        private int b;

    }
}
```

Anonymous classes

We have a method:

```
public void giveMeYourAge(MyAge a) {
    //...
}
```

Which takes an object inheriting:

```
public interface MyAge {
    public int age();
}
```

How do we give it an age of 21?

Anonymous classes

We have a method:

```
public void giveMeYourAge(MyAge a) {
    //...
}
```

Which takes an object inheriting:

```
public interface MyAge {
    public int age();
}
```

How do we give it an age of 21?

1: Create a new subclass

```
public class MyClass implements MyAge {

    public int age() {
        return 21;
    }
}
```

```
MyClass mc = new MyClass();
giveMeYourAge(mc);
```

Anonymous classes

We have a method:

```
public void giveMeYourAge(MyAge a) {
    //...
}
```

Which takes an object inheriting:

```
public interface MyAge {
    public int age();
}
```

How do we give it an age of 21?

2: Create an anonymous class

```
giveMeYourAge(new MyAge() {
    public int age() {
        return 21;
    }
});
```

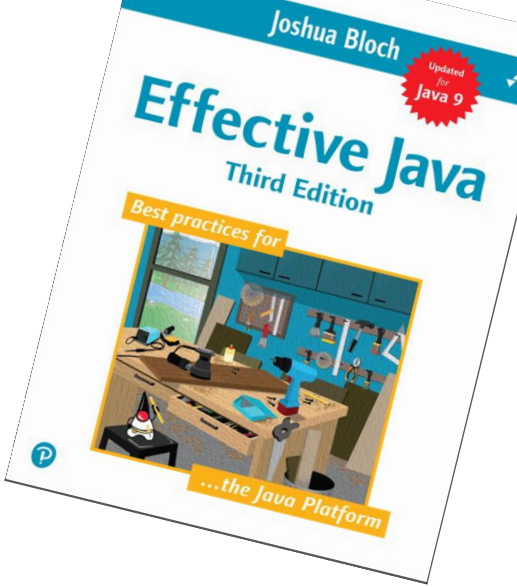
Sidenotes: this and super

```java
public class Circle {

    private double radius;

    public Circle(double radius) {
        radius = radius;
        this.radius = radius;
    }

}
```

```java
class A {
    public int value;
}

class B extends A {
    public int value;

    public int getA() {
        return super.value;
    }

    public int getB() {
        return value;
    }
}
```

Bonus! Last week review

Equals Methods



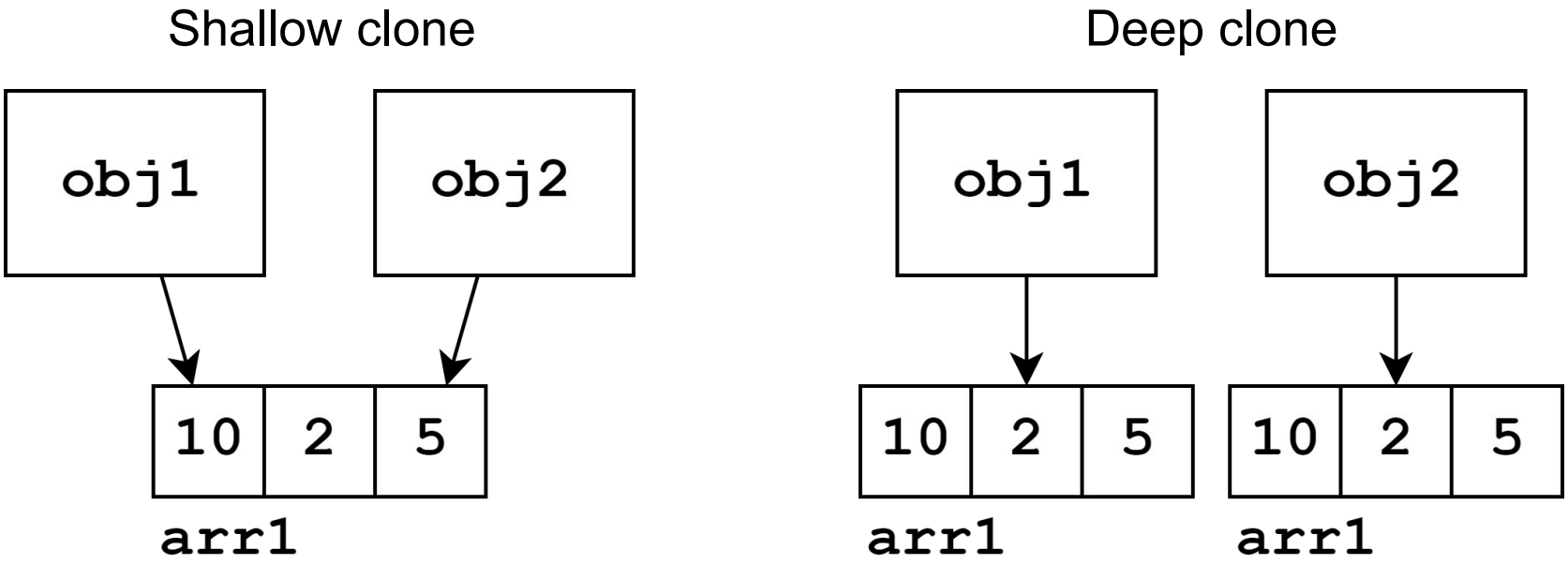
Find the book at this link: https://the-eye.eu/public/Books/IT%20Various/Effective%20Java%2C%202nd%20Edition.pdf

==	equals()
● Can be used for primitives and objects ● Compares primitives values ● Compares object locations	● Can only be used for objects ● All objects implement it (Default: compare locations) ● **Can be overwritten for a new class to compare fields**

Clone Method

Find the book at this link: https://the-eye.eu/public/Books/IT%20Various/Effective%20Java%2C%202nd%20Edition.pdf

Shallow vs. Deep Clone



Future material: Big-O

Big-O Big-Misconception

Big-O does **NOT** describe how fast your algorithm is. It is a measure of the rate your runtime increases as input size increases.

```java
//O(n)
public int fast(int n) {

    int k = 1;
    for(int i = 2; i < n; i++) {
        k = k * i;
    }

    return k;
}
```

```java
//O(n)
public int slow(int n) throws Exception {

    int k = 1;
    for(int i = 2; i < n; i++) {
        k = k * i;
        Thread.sleep(100000000);
    }

    return k;
}
```