THE UNIVERSITY OF MELBOURNE

Lab6（week9）

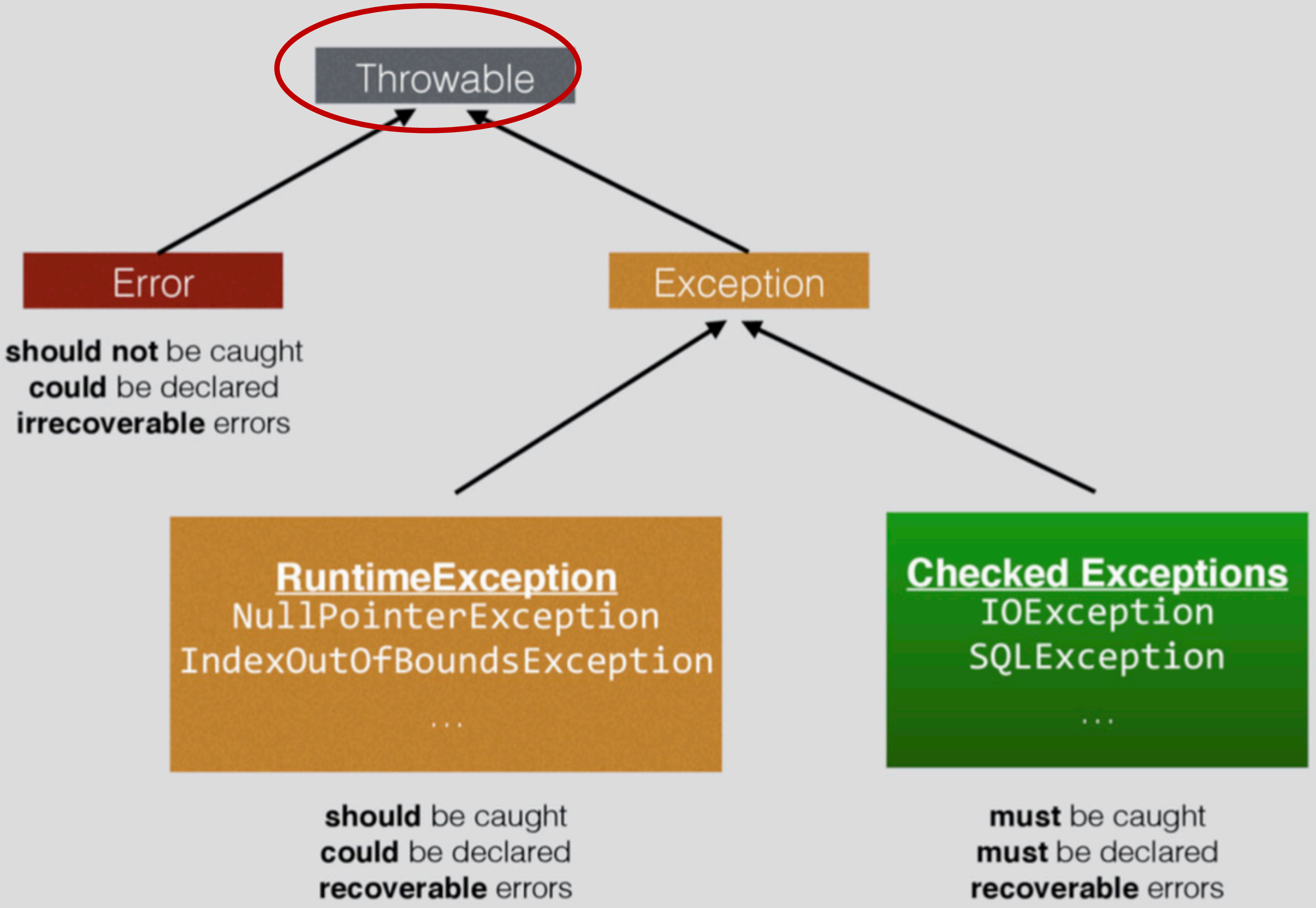
COMP90041 Programming

and software development

Zhe(Zoe) Wang

github: https://github.com/Zoeewang/COMP90041-2020-sem1-tutorial

The Java Exception Hierarchy



Exception

- An exception is an object indicating what went wrong

- methods of all exception classes:

- toString(): return a String describing the exception

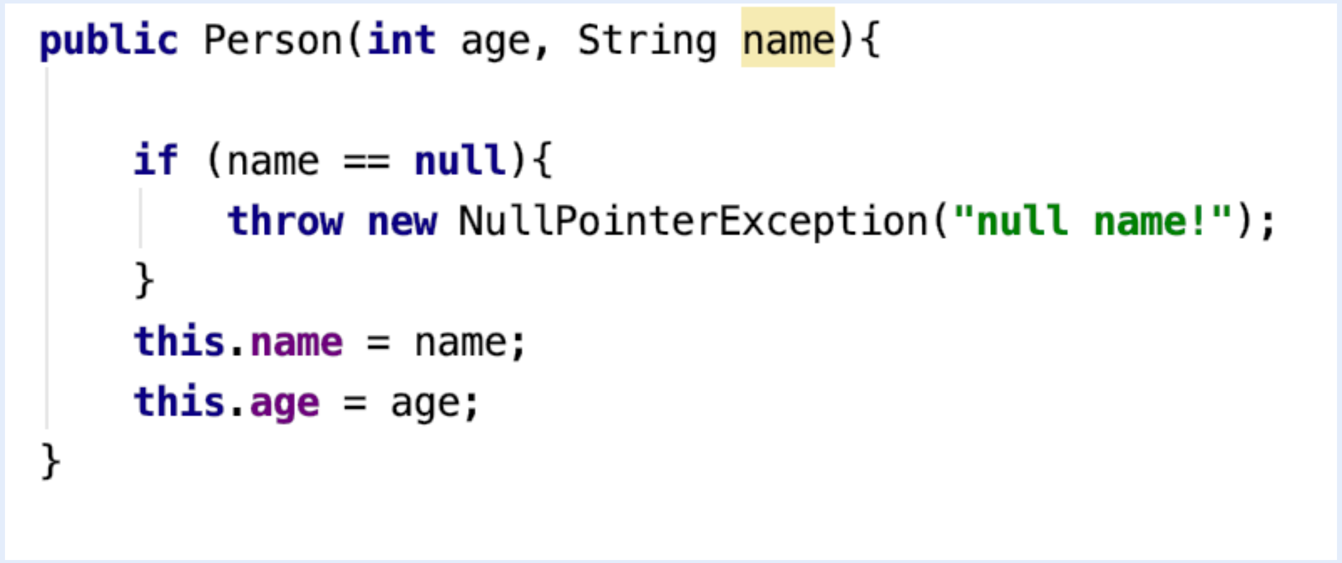
- getMessage(): returns a string with detail about the error

- printStackTrace(): print a backtrace of what was happening(only useful to programmers

Throwing

- currently executing code is interrupted
- if never caught, the program is aborted, print backtrace

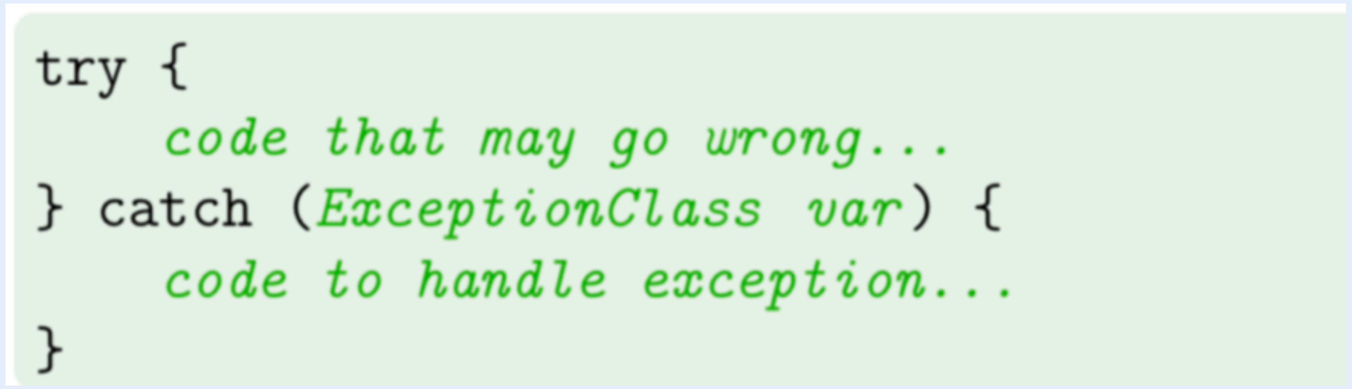
form: throw new *exceptionClass(detail String);*



```
public Person(int age, String name){

    if (name == null){
        throw new NullPointerException("null name!");
    }
    this.name = name;
    this.age = age;
}
```

Handling Exceptions



```
try {
    code that may go wrong...
} catch (ExceptionClass var) {
    code to handle exception...
}
```

try: specifies code that may throw an exception

catch para: specifies the kind of exception to catch

inside catch: what to do if exception occurs

demo1

Catching

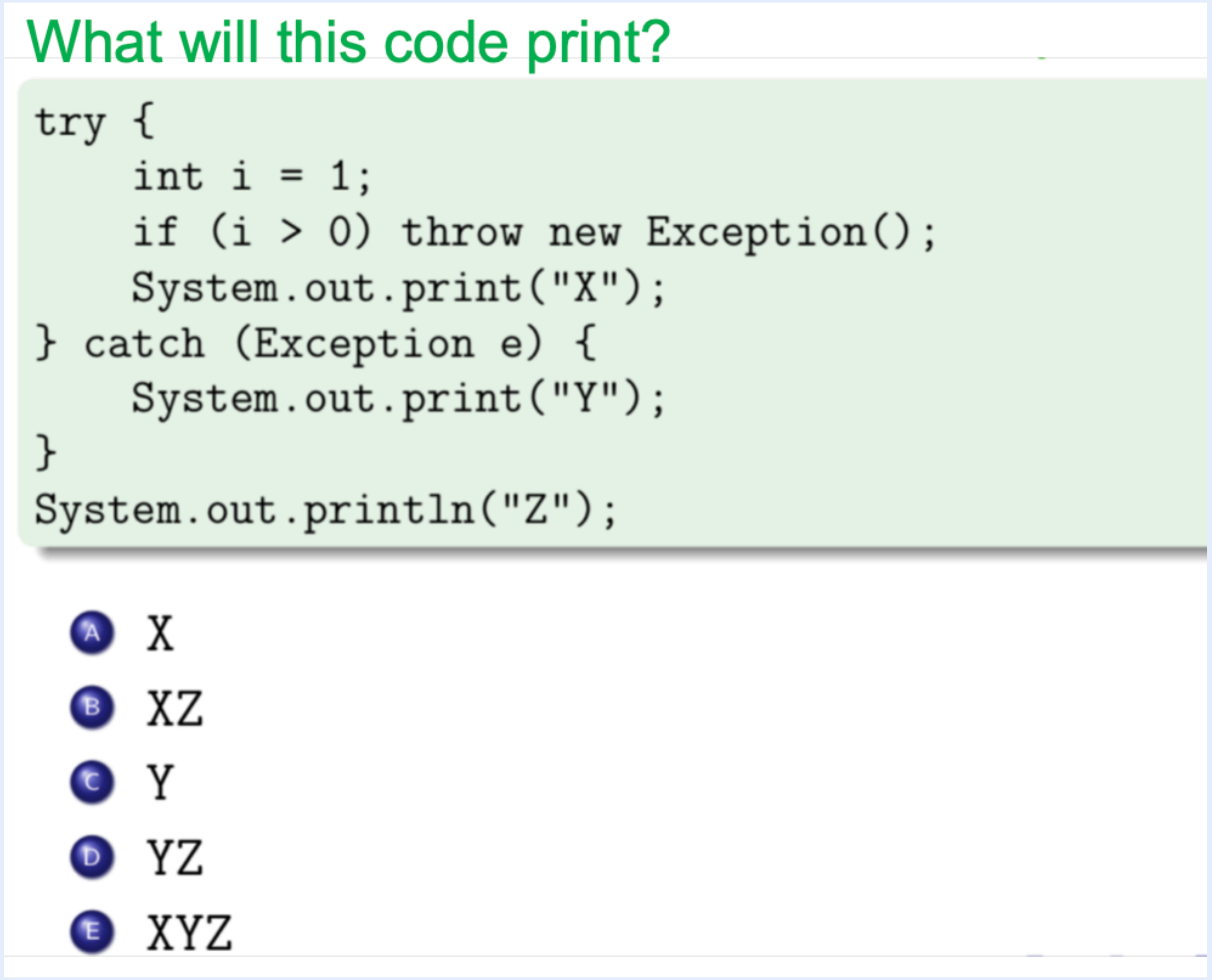
catch para: catch(NullPointerException e)

multiple catches:

- only one handler is executed, others are ignored
- first one that matches the thrown exception is used

Always put more specific catches before general ones

demo2



What will this code print?

```
try {
    int i = 1;
    if (i > 0) throw new Exception();
    System.out.print("X");
} catch (Exception e) {
    System.out.print("Y");
}
System.out.println("Z");
```

- A X
- B XZ
- C Y
- D YZ
- E XYZ

Catch

- try recover from error

- e.getMessage() returns exception message

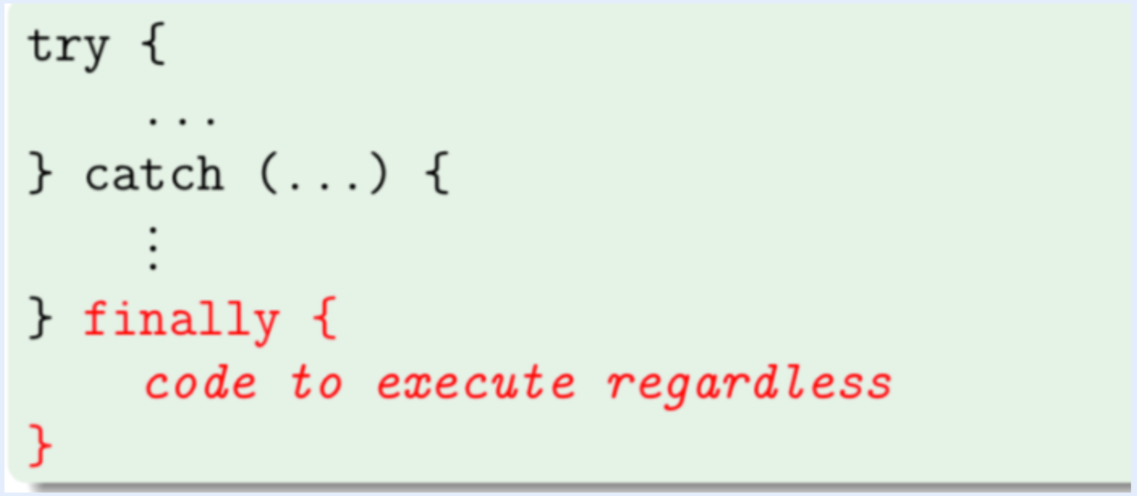
- cannot resolve - throw same exception (e)

- throw a different exception

- only exception thrown inside the try block are caught by that try…catch

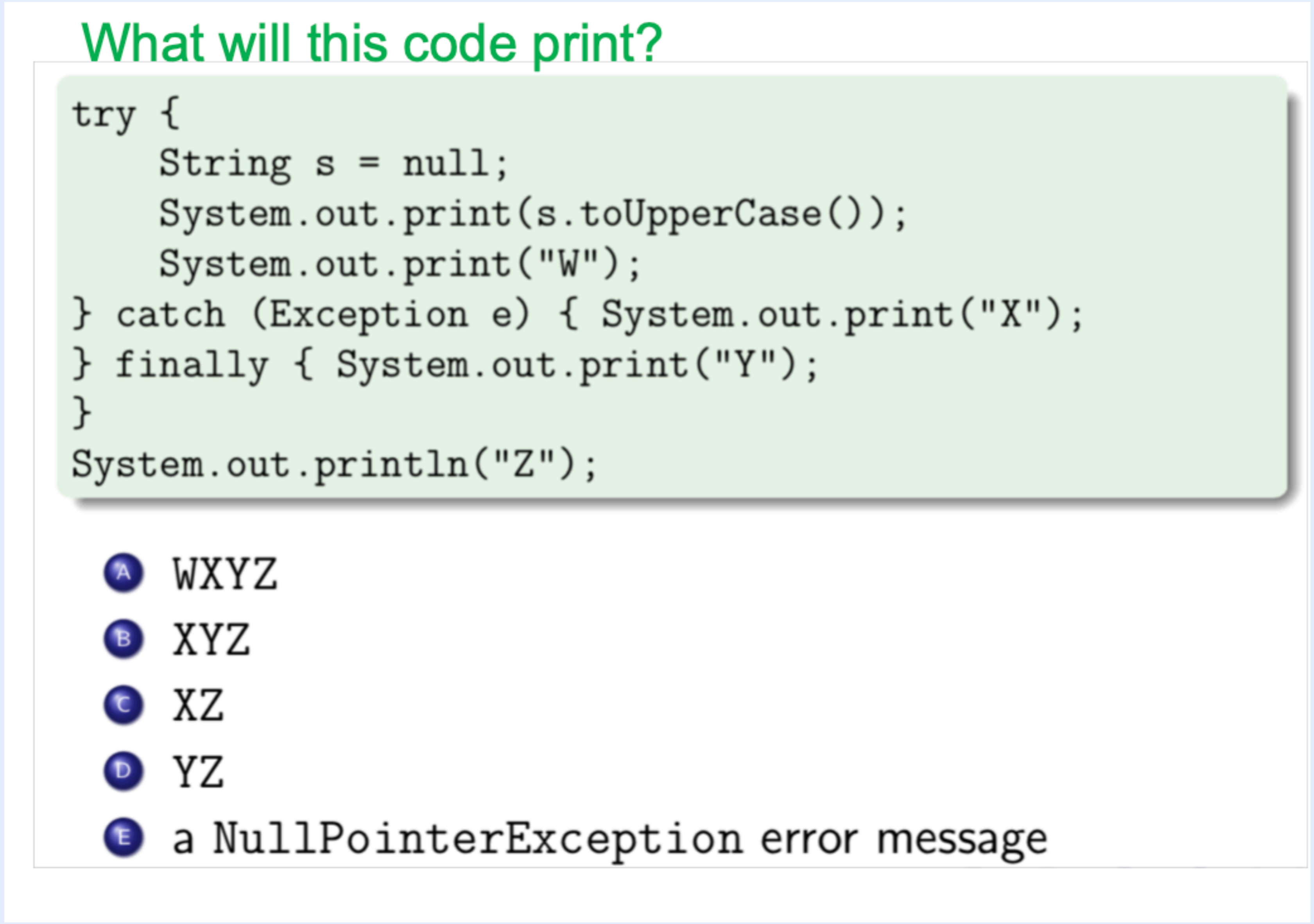
- demo3

finally



```
try {
    ...
} catch (...) {
    ⋮
} finally {
    code to execute regardless
}
```

- **finally block is executed almost no matter what**
- **only if try or catch is an infinite loop or calls System.exit, finally missed**
- **demo4**



What will this code print?

```
try {
    String s = null;
    System.out.print(s.toUpperCase());
    System.out.print("W");
} catch (Exception e) { System.out.print("X");
} finally { System.out.print("Y");
}
System.out.println("Z");
```

- A WXYZ
- B XYZ
- C XZ
- D YZ
- E a NullPointerException error message

throws

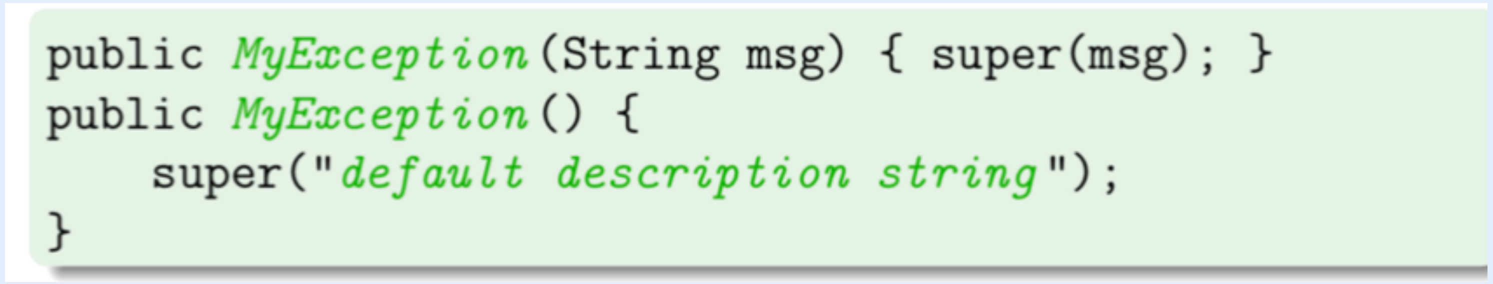
declare what checked exceptions a method can throw with a throws

throws *ExceptionClass*

Define exceptions

- must be descendent of **Exception** class

- Usually define a constructor with no arguments and one with a single String argument



```
public MyException(String msg) { super(msg); }
public MyException() {
    super("default description string");
}
```

Writing to a text file

The class PrintWriter is a stream class that can be used to write to a text file

```
import java.io.PrintWriter;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;
```

```
PrintWriter outputStream = null;
String filename = "out.txt";
outputStream = new PrintWriter(new FileOutputStream(filename, true));
```

```
outputStreamName.close();
```

IOException

FileNotFoundException

IOException

-use try...catch

Write Object (in binary)

```
import java.io.ObjectOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

ObjectOutputStream outputStream = null;
String filename = "dogs.dat";

outputStream = new ObjectOutputStream(new FileOutputStream(filename));
outputStream.writeObject(dogs);
```

Read Object

```
import java.io.ObjectInputStream;
import java.io.IOException;
import java.io.FileInputStream;

ObjectInputStream inputStream = null;
String filename = "dogs.dat";

inputStream = new ObjectInputStream(new FileInputStream(filename));
dogs =(Dog[]) inputStream.readObject();
```

type cast!

ClassNotFoundException

Binary I/O of Object

In addition, the class of the object being read or written must implement the Serializable interface

```
public class Dog implements Serializable {...}
```

Q5

Consider a graphics system that has classes for various figures—say, rectangles, boxes, triangles, circles, and so on. For example, a rectangle might have data members' height, width, and center point, while a box and circle might have only a center point and an edge length or radius, respectively. In a well-designed system, these would be derived from a common class, Figure. You are to implement such a system.

The class Figure is the base class. You should add only Rectangle and Triangle classes derived from Figure. Each class has stubs for methods erase and draw. Each of these methods outputs a message telling the name of the class and what method has been called. Because these are just stubs, they do nothing more than output this message. The method center calls the erase and draw methods to erase and redraw the figure at the center. Because you have only stubs for erase and draw, center will not do any "centering" but will call the methods erase and draw, which will allow you to see which versions of draw and center it calls. Also, add an output message in the method center that announces that center is being called. The methods should take no arguments. Also, define a demonstration program for your classes.

Q6

Flesh out Programming Project 8.5. Give new definitions for the various constructors and methods center, draw, and erase of the class Figure; draw and erase of the class Triangle; and draw and erase of the class Rectangle. Use character graphics; that is, the various draw methods will place regular keyboard characters on the screen in the desired shape. Use the character '*' for all the character graphics. That way, the draw methods actually draw figures on the screen by placing the character '*' at suitable locations on the screen. For the erase methods, you can simply clear the screen (by outputting blank lines or by doing something more sophisticated). There are a lot of details in this project and you will have to decide on some of them on your own.

Q7

Define a class named MultiItemSale that represents a sale of multiple items of type Sale given in Display 8.1 (or of the types of any of its descendent classes). The class MultiItemSale will have an instance variable whose type is Sale[], which will be used as a partially filled array. There will also be another instance variable of type int that keeps track of how much of this array is currently used. The exact details on methods and other instance variables, if any, are up to you. Use this class in a program that obtains information for items of type Sale and of type DiscountSale (Display 8.2) and that computes the total bill for the list of items sold.

THE UNIVERSITY OF MELBOURNE

Thank you