

## **WES-CS GROUP MEETING #13**

### **Exercise 1: Practice with Exceptions**

Divide into two teams to play a game like Jeopardy involving questions about the code on the next page. For each question, the team that rings their bell first gets to try to answer the question. If the answer is right, the team gets points, and gets to choose the point value of the next question (100, 200, or 300 points).

```

public class ExceptionExample {

    public static void main(String[] args) {
        foo();
        try {
            bar();
        } catch (IndexOutOfBoundsException iobe) {
            System.out.println("ERROR 1");
        } catch (ArithmeticException ae) {
            System.out.println("ERROR 2");
        } finally {
            System.out.println("DONE");
        }
    }

    public static void foo() {
        // PROGRAM LINE A
        try {
            // PROGRAM LINE B
        } catch (NullPointerException npe) {
            System.out.println(npe.getMessage());
            npe.printStackTrace();
        }
    }

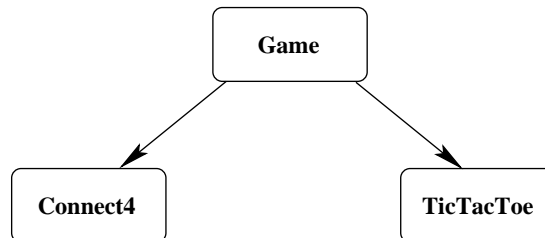
    public static void bar() {
        // PROGRAM LINE C
        try {
            // PROGRAM LINE D
            methodX();
        } catch (NullPointerException npe) {
            System.out.println("ERROR 3");
        } catch (IndexOutOfBoundsException iobe) {
            System.out.println("ERROR 4");
        }
        System.out.println("DONE BAR");
    }

    public static void methodX() {
        // PROGRAM LINE E
        try {
            // PROGRAM LINE F
        } catch (IndexOutOfBoundsException iobe) {
            System.out.println(iobe.getMessage());
            iobe.printStackTrace();
        } catch (NumberFormatException nfe) {
            System.out.println("ERROR 5");
            return;
        } finally {
            System.out.println("DONE METHODX");
        }
    }
}

```

## Exercise 2: Writing a Game class hierarchy

Last week you worked on designing and writing a program that lets two people play a game of Connect-4 against each other on the computer. Today we're going to think about how to make the *Connect4* class fit into a class hierarchy that looks like this:



A reasonable set of methods for the *Connect4* class is given on the next page. Use those, or use the methods you wrote last week (whichever you prefer). Decide which *Connect4* methods are also good methods to have in the *TicTacToe* class. Which of those methods can be exactly the same, and which should have the same signatures but different implementations?

Now write the *Game* class. It should include all of the methods that you decided should be in both the *Connect4* and *TicTacToe* classes. The methods that can be exactly the same in both of those classes should be moved from the *Connect4* class to the *Game* class (because then the *Connect4* and the *TicTacToe* classes will inherit those methods, so you won't have to write them twice). The methods that need different implementations should also be declared in the *Game* class, but only their signatures should be given (not the method bodies), and they should be *abstract* methods (that means that you start the method declaration with the keyword *abstract*). When a class contains an abstract method, the class itself must also be declared abstract, and every subclass must include an implementation of the abstract methods. (An abstract class is similar to an *interface*: both include declarations of method signatures that must be implemented in every class that *extends* the abstract class, and in every class that *implements* the interface).

Change your *Connect4* class so that it is a subclass of the *Game* class. Make sure that it doesn't redefine the non-abstract methods in the *Game* class, but does define the abstract ones.

Finally, write a *TicTacToe* class that is also a subclass of the *Game* class, and that allows two people to play a game of tic-tac-toe against each other on the computer.

<pre>void play() void doNextMove()  boolean isBadMove(int move)  void doOneMove(int move) void printBoard() boolean gameOver()  boolean boardFull()  boolean hasWon(int playerNum)  boolean hasPattern   ( int[][] pattern )  boolean hasPatternIgnoreZeros   ( int[][] pattern )  boolean isMatch   (int[][] pattern, int row, int col)  boolean isMatchIgnoreZeros   (int[][] pattern, int row, int col)</pre>	<p>play one game</p> <p>handle getting, verifying, and recording the next player's move</p> <p>return true iff the given move is not legal</p> <p>given a legal move, record it</p> <p>print the current gameboard</p> <p>return true iff the game is over</p> <p>return true iff the whole gameboard is full</p> <p>return true iff the given player has won</p> <p>return true iff the given pattern occurs in the gameboard</p> <p>return true iff the given pattern occurs in the gameboard, where zeros in the pattern can match any value in the gameboard</p> <p>return true iff the given pattern occurs in the gameboard with its upper-left corner in the given position</p> <p>return true iff the given pattern occurs in the gameboard with its upper-left corner in the given position, where zeros in the pattern can match any value in the gameboard</p>
--	---

### Exercise 3: Practice with Inheritance

Below are six class definitions (ParentThing, ThingOne, ThingTwo, ThingThree, ChildOne, and ChildTwo). Remember that “extends” indicates inheritance. For instance, “ThingOne extends ParentThing” means that ThingOne is a subclass of ParentThing, and so ThingOne has all the data members and methods of ParentThing, plus some of its own.

```
class ParentThing {
    public int j;
    protected int k;
    private int p;
    public void methodA() { // Code segment 1 }
    public void methodA(String str) { // Code segment 2 }
    public void methodB() { // Code segment 3 }
}

class ThingOne extends ParentThing {
    public int onej;
    protected int onek;
    private int onep;
    public void methodA(int c) { // Code segment 4 }
}

class ThingTwo extends ParentThing {
    public void methodB() { // Code segment 5 }
    public void methodB(String str) { // Code segment 6 }
}

class ThingThree extends ParentThing {
    public void methodB() { // Code segment 7 }
}

class ChildOne extends ThingOne {
    public void methodA() { // Code segment 8 }
}

class ChildTwo extends ThingTwo {
    public void methodA() { // Code segment 9 }
}
```

Assume that the following statements have been executed.

```
ParentThing parent = new ParentThing();
ThingOne one = new ThingOne();
ThingTwo two = new ThingTwo();
ThingThree three = new ThingThree();
ChildOne kid1 = new ChildOne();
ChildTwo kid2 = new ChildTwo();
```

Divide into two teams to play a game like Jeopardy with 100, 200, and 300-point questions about the code above and on the previous page. For each question, the team that rings the bell first gets to try to answer the question. If the answer is right, the team gets points, and gets to choose the point value of the next question.

Some questions involve method calls made from *outside* the six classes (using the variables defined above). For each, you must say whether the code compiles without error, and if it does compile, which code segment in the class definitions given on the previous page executes when the method is called.

Other questions involve replacements for the code segments labeled

```
// Code segment 1
// Code segment 2
```

etc. For each, you must say whether the statement will cause a compile-time error and if so why.

## 100 point questions for Exceptions

Q: What happens if *throw new NumberFormatException()* replaces line A?

A: A stack trace would be printed, and the program would crash.

Q: What happens if *throw new NumberFormatException()* replaces line B?

A: A stack trace would be printed, and the program would crash.

Q: What happens if *throw new NumberFormatException()* replaces line C?

A: DONE BAR would be printed, then a stack trace, and the program would crash.

Q: What happens if *throw new NumberFormatException()* replaces line D?

A: DONE BAR would be printed, then a stack trace, and the program would crash.

Q: What happens if *throw new NullPointerException()* replaces line C?

A: DONE BAR would be printed, then a stack trace, and the program would crash.

Q: What happens if *throw new NullPointerException()* replaces line D?

A: The following would be printed:

```
ERROR 3
DONE BAR
DONE
```

Q: What happens if *throw new IndexOutOfBoundsException()* replaces line A?

A: A stack trace would be printed, and the program would crash.

Q: What happens if *throw new IndexOutOfBoundsException()* replaces line B?

A: A stack trace would be printed, and the program would crash.

Q: What happens if *throw new ArithmeticException()* replaces line A?

A: A stack trace would be printed, and the program would crash.

Q: What happens if *throw new NullPointerException()* replaces line A?

A: A stack trace would be printed, and the program would crash.

## 200 point questions for Exceptions

- Q: What happens if *throw new ArithmeticException()* replaces line B?  
A: A stack trace would be printed, and the program would crash.
- Q: What happens if *throw new ArithmeticException()* replaces line C?  
A: The following would be printed:   ERROR 2  
  DONE
- Q: What happens if *throw new ArithmeticException()* replaces line D?  
A: The following would be printed:  
      ERROR 2  
      DONE
- Q: What happens if *throw new IndexOutOfBoundsException()* replaces line C?  
A: The following would be printed:  
      ERROR 1  
      DONE
- Q: What happens if *throw new IndexOutOfBoundsException()* replaces line D?  
A: The following would be printed:  
      ERROR 4  
      DONE BAR  
      DONE
- Q: What happens if *throw new NumberFormatException()* replaces line E?  
A: DONE METHODX would be printed, then a stack trace, then the program would crash
- Q: What happens if *throw new NullPointerException()* replaces line B?  
A: The NullPointerException message would be printed, then a stack trace, then the following:  
      DONE METHODX  
      DONE BAR  
      DONE
- Q: What happens if *throw new NullPointerException()* replaces line E?  
A: The following would be printed:  
      ERROR 3  
      DONE BAR  
      DONE
- Q: What happens if *throw new NullPointerException()* replaces line F?  
A: The following would be printed:

**300 point questions for Exceptions**

DONE METHODX  
ERROR 3  
DONE BAR  
DONE

### 300 point questions for Exceptions

Q: What happens if *throw new IndexOutOfBoundsException()* replaces line E?

A: The following would be printed:

```
ERROR 4  
DONE BAR  
DONE
```

Q: What happens if *throw new IndexOutOfBoundsException()* replaces line F?

A: The `IndexOutOfBoundsException` message would be printed, then the stack trace, then:

```
DONE METHODX  
DONE BAR  
DONE
```

Q: What happens if *throw new ArithmeticException()* replaces line E?

A: The following would be printed:

```
ERROR 2  
DONE
```

Q: What happens if *throw new ArithmeticException()* replaces line F?

A: The following would be printed:

```
DONE METHODX  
ERROR 2  
DONE
```

Q: What happens if *throw new NumberFormatException()* replaces line F?

A: The following would be printed:

```
ERROR 5  
DONE METHODX  
DONE BAR  
DONE
```

Q: What happens if no lines are replaced with code that throws exceptions?

A: The following would be printed:

```
DONE METHODX  
DONE BAR  
DONE
```

Q: Replacing what lines with *throw new IndexOutOfBoundsException()* would cause message "ERROR 1" to be printed (ignore other things that might be printed as well)?

A: Line C only

### 100 point questions for Inheritance

Q: Does `ParentThing.methodA();` compile; if yes, which code segment executes?

A: Does not compile (there is no static `methodA` in the `ParentThing` class).

Q: `parent.methodB("Testing");`

A: Does not compile (there is no `methodB` with a `String` parameter in the `ParentThing` class).

Q: `parent.methodB();`

A: Compiles and executes Code segment 3.

Q: `two.methodB();`

A: Compiles and executes Code segment 5.

Q: `two.methodA();`

A: Compiles and executes Code segment 1.

Q: `kid2.methodA();`

A: Compiles and executes Code segment 9.

Q: `ThingOne.methodB();`

A: Does not compile (there is no static `methodB` in the `ThingOne` class).

Q: `// Replacement for code segment 4  
j = onej;`

A: OK (`j` is a public field of the parent class, `onej` is a field of this class)

Q: `// Replacement for code segment 4  
methodB();`

A: OK (parent class `ParentThing` has a `methodB` with no parameter)

## 200 point questions for Inheritance

Q: `kid1.methodB();`

A: Compiles and executes Code segment 3.

Q: `// Replacement for code segment 4  
methodB( "hello" );`

A: NO (neither ThingOne nor parent class have methodB with a String parameter)

Q: `kid1.methodA(45);`

A: Compiles and executes Code segment 4.

Q: `one.methodA( "5" );`

A: Compiles and executes Code segment 2.

Q: `kid1.methodA( "Smile" );`

A: Compiles and executes Code segment 2

Q: `kid2.methodA( "Hello World!" );`

A: Compiles and executes Code segment 2

Q: `three.methodA(10);`

A: Does not compile (neither ThingThree nor ParentThing has a methodA with an int parameter).

Q: `three.methodB( "String" );`

A: Does not compile (neither ThingThree nor ParentThing has a methodB with a String parameter).

Q: `// Replacement for code segment 9  
k++;`

A: OK (k is a protected field of a superclass)

### 300 point questions for Inheritance

Q: `// Replacement for code segment 4`  
`k = onek;`

A: OK (k is a protected field of the parent class, onek is a field of this class)

Q: `// Replacement for code segment 4`  
`k = onep;`

A: OK (k is a protected field of the parent class, onep is a field of this class)

Q: `// Replacement for code segment 4`  
`p = onep;`

A: NO (p is a *private* field of the parent class)

Q: `// Replacement for code segment 8`  
`j = onej;`

A: OK (j and onej are public fields of superclasses)

Q: `// Replacement for code segment 8`  
`k = onek;`

A: OK (k and onek are protected fields of superclasses)

Q: `// Replacement for code segment 8`  
`onep++;`

A: NO (onep is a *private* field of the parent class)

Q: `// Replacement for code segment 8`  
`methodA(22);`

A: OK (parent class has methodA with an int parameter)

Q: `// Replacement for code segment 9`  
`j = onej;`

A: NO (onej is a field of ThingOne, which is not a superclass)