

WES-CS GROUP MEETING #14

Exercise 1: The Performing Arts (Interfaces and Inheritance)

Part (a). A performer has one or more of the following skills: acting, singing, dancing. A performer can tell you, for each of the three skills, whether they have that skill or not. A performer will also perform all of their skills on demand: that is, they will return a string that says what skills they're performing. For example, a performer who has both signing and acting skills would return the string "I'm singing and acting".

Write a `Performer` interface that defines this functionality.

Part (b). Assume that the following `Person` class has been defined.

```
public class Person {
    private String name;
    public Person(String aName) { name = aName; }
    public String getName()      { return name; }
}
```

An opera singer is a person who is also a performer, and who earns a salary. Every opera singer can sing. Some opera singers can also act (and that is specified when the constructor is called), but no opera singers can dance. An opera singer's salary is set when the singer is created. The salary can be both accessed and modified.

Write the complete `OperaSinger` class definition, including appropriate fields, a constructor, and the required methods.

Part (c). Now assume that you have four variables, `p1`, `p2`, `p3`, and `p4`, that represent people who are different kinds of performers. For example, the variables might have types `OperaSinger`, `BalletDancer`, `Thespian`, and `TVActor`. Write a code fragment to allocate an array of `Performer`, initialize the array with those four variables, and then print the names of each performer in the array and have them perform their skills on demand.

Part (d). Would it make sense to make `Performer` a subclass of `Person` instead of an interface? How would that work?

Exercise 2: Coffee Tea or Milk? (Practice with Inheritance)

The code shown below defines three classes: Drink, CocaCola, and Milk. Coca-Cola and Milk are each subclasses of Drink.

```
public class Drink {
    private double myPrice;
    public Drink(double price) {
        myPrice = price;
    }
    public double getPrice() {
        return myPrice;
    }
    public String getName() {
        return "Drink";
    }
    public void printLabel() {
        System.out.println( getName() );
        System.out.println("Price: " + getPrice());
    }
}

public class CocaCola extends Drink {
    private static final double PRICE = 1.25;
    public CocaCola() {
        super(PRICE);
    }
    public String getName() {
        return "Coca-Cola Classic";
    }
}

public class Milk extends Drink {
    private static final double PRICE = .75;
    private int percentFat; // either 1 or 2
    public Milk( int fat ) {
        super(PRICE);
        percentFat = fat;
    }
    public String getName() {
        return "Milk";
    }
    public int getFat() {
        return percentFat;
    }
}
```

Your job is to complete the `DrinkTester` class below by writing the `getDrink` method and adding the code in the `main` method that prints the fat content if the drink is milk.

```
import java.util.*;

public class DrinkTester {
    // PART (a): WRITE THE getDrink METHOD HERE
    public static void main(String[] args) {
        for (int k=1; k<6; k++) {
            Drink drink = getDrink();
            drink.printLabel();
            // PART (b): ADD CODE HERE TO PRINT THE FAT CONTENT
            // IF drink IS MILK
        }
    }
}
```

Part (a)

The `getDrink` method should create and return a `Drink`, randomly choosing between `CocaCola` and `Milk`. If it chooses `Milk`, it should randomly choose between 2% and 1% for the fat content. Before writing the code, think about the following questions:

- Should the `getDrink` method be a *static* or a *non-static* method?
- What are the possible types that will actually be returned by the `getDrink` method?
- What should the return type of the `getDrink` method be?
- When `printLabel` is called (just after the call to `getDrink`), it causes a call to `getName`. There are three versions of that method, defined in the `Drink`, `Coca-Cola`, and `Milk` classes. What determines which version is actually called?

Now write the code for the `getDrink` method.

Part (b)

What code could we add at the end of the `main` method that would figure out whether `drink` is a `Milk` object, and if so, would print its fat content?

Exercise 3: Let's Play Catch! (Exceptions)

Sometimes when things go wrong in your code, an exception is thrown. For example, you may have encountered an `IndexOutOfBoundsException` when your code goes beyond the bounds of an array, or a `NullPointerException` when you try to use an uninitialized object. Exceptions are used to communicate when something goes wrong in a program and to indicate what has gone wrong so the program can try to recover from the problem. In this exercise we're going to learn more about exceptions and how they work in Java.

Some statements (like those that access elements of arrays or those that dereference pointers) can cause an exception to be thrown. Another way to cause an exception to be thrown is by writing a `throw` statement. For example

```
throw new GreenEx();
```

will create an `GreenEx` object (which must be a subclass of the `Exception` class), and will throw that exception. When an exception is thrown in a method, it must either be caught inside that method or it will be thrown again at the place where the method was called. If an exception is ever thrown but not caught in the `main` method, the program crashes (and displays a message about the uncaught exception).

Methods handle exceptions with `try/catch` statements. The idea is that you *try* some code that may throw an exception, and you provide code to *catch* some of those exceptions if they are indeed thrown.

Assume that classes `GreenEx`, `BlueOrangeEx`, `RainbowEx` have been defined (all with no-argument constructors). Consider the code on the next page.

```

1.  public static void main( String [] args ) {
2.      System.out.println("start main");
3.      try {
4.          foo();
5.          bar();
6.          fraz();
7.      } catch( BlueOrangeEx ex ) {
8.          System.out.println("main caught BlueOrangeEx");
9.      } catch( RainbowEx ex ) {
10.         System.out.println("main caught RainbowEx");
11.     }
12.     System.out.println("end main");
13. }

14. public void foo() {
15.     System.out.println("start foo");
16.     try {
17.         bar();
18.     } catch ( BlueOrangeEx ex ) {
19.         System.out.println("foo caught BlueOrangeEx");
20.     }
21.     bar();
22.     System.out.println("end foo");
23. }

24. public void bar() {
25.     System.out.println("start bar");
26.     try {
27.         fraz();
28.     } catch ( GreenEx ex ) {
29.         System.out.println("bar caught GreenEx");
30.         throw new BlueOrangeEx();
31.     }
32.     System.out.println("end bar");
33. }

34. public void fraz() {
35.     System.out.println("start fraz");
36.     try {
37.         // Team Leader says which exception to throw here
38.     } catch ( RainbowEx ex ) {
39.         System.out.println("fraz caught RainbowEx");
40.     }
41.     System.out.println("end fraz");
42. }

```

Now we'll act out the code as follows.

- Start with one person standing up and simulating the execution of `main`.
- When you get to a print statement, just say what is printed.
- When you get to a method call, the person next to you will stand and simulate that method.
- When your method finishes, sit down.
- When method `fraz` enters its `try` block, your Team Leader will give that person a `GreenEx`, `BlueOrangeEx`, or `RainbowEx` to throw.
- Exceptions should be passed up the call chain and caught by the appropriate person: that is, the person simulating the method with the closest enclosing `try/catch` block that has a `catch` for the thrown exception. If there is no such person, then the exception should fall to the floor.

Repeat this exercise a few times. Your Team Leader will choose different exceptions for `fraz` to throw.

Exercise 4: Hats and Snacks (Logical Thinking)

Puzzle 1: Four people are standing in a row, one behind the other, all facing the same way with person 1 at the front and person 4 at the end. Between numbers 1 and 2 is a wall that prevents any of the others from seeing person 1. So number 4 can see numbers 3 and 2, number 3 can see number 2, and numbers 2 and 1 cannot see anyone.

Problem: The people are told that two of them are wearing white hats and two are wearing black hats. They are told that if any one of them can say what color hat they are wearing, they will all win fabulous prizes! They are not allowed to communicate with each other in any way. Will they be able to win the fabulous prizes? If so, why?

Puzzle 2: Remember the nasty dictator who liked to play games with his prisoners? He's at it again! This time, he tells the prisoners that the next day they will be lined up so that person 1 sees nothing, person 2 sees person 1, person 3 sees people 1 and 2, etc. Each of them will have either a black or a white hat put on their head. Then the last person in line will be asked what color his own hat is. If he gets it right, he's free. If he gets it wrong, he's dead. Either way, the question is then posed to the second-to-last person in line. And so on.

Problem: How can the prisoners plan to answer the questions so as to maximize the number of lives saved?

Puzzle 3: During the lunch hour at school, a group of five boys from Miss Smith's home-room visited a nearby snackbar. One of the five boys took a candy bar without paying for it. When the boys were questioned by the school principal, they made the following statements:

Rex: "Neither Earl nor I did it."

Jack: "It was Rex or Abe."

Abe: "Both Rex and Jack are lying."

Dan: "Abe's statement is not true."

Earl: "What Dan said is wrong."

When Miss Smith was consulted, she said, "Three of these boys always tell the truth, and the other two always lie." Assuming that Miss Smith is correct, determine who took the candy bar.