Object-Oriented programming and design

Laboratory work #3 (5 pts)

Interfaces

Deadline: week 11

#1

When to use an Interface vs when to use an abstract class.

For each "when" provide extended example(s) (with class/interface codes).

#2

Suppose you have an interface `Moveable`.

Think of some interface that can extend it and create it.

Implement this two interfaces in some classes.

#3

A collection represents a group of objects, known as its elements. Some collections allow duplicate elements and others do not. Some are ordered and others unordered.

Create an Interface `MyCollection` which is **maximum** general (abstract) collection possible.

#4

Create a class called `Employee` whose objects are records for an employee. This class will be a derived class of the class `Person`.

An employee record has an employee's `name` (inherited from the class Person), an annual `salary` represented as a single value of type double, a `hireDate` of type `java.util.Date` (date the employee started work) and a national `insuranceNumber`, which is a value of type String. Inside this class you need to override `toString` and `equals` methods of the `Person` class. Your class should have a reasonable number of constructors and accessor methods.

Then create a class `Manager` extending `Employee`, each manager has a team of Employees (Vector) and can get a bonus. You need to override `toString` and `equals` methods.

Next, implement `Comparable` interface. (Employee1 > Employee2 if its `salary` is more than the salary of Employee2, the same for managers, but if their salaries are equal, compare by `bonus`). In addition, provide 2 comparators – one to sort by name and the other to sort by hire date.

Finally, write a `clone`() method for the Employee and Manager classes. Use shallow or deep cloning as you want.

Write another class containing a main method to fully test your class definition. Check that your implementations are working fine.

#5

Write a class `Chocolate` with fields `weight` and `name` (e.g., Twix) and method `toString()`. Implement **Comparable** interface (chocolates must be compared by `weight`).

Implement **Comparable** interface for a `Time` class you created during week 2. Then import this class to your current project.

Next, implement a `Sort` class that will be able to sort anything that can be compared to each other (e.g. anything that is **Comparable**). You have to have a `swap` method that is commonly used in sorting algorithms (this ensures that you will not duplicate this code each time you need to swap elements) and 2 methods for sorting. USE ANY SORTING ALGORITHMS YOU LIKE. Bubble Sort and Merge Sort are given just for example.

```
public class Sort {
     static <E> void swap(E [] array, int i, int j) {
        ///..
    }
//this means that E is an arbitrary data type

     static <E extends Comparable<E>> void bubbleSort(E []
array) {
       ///…
/////this means that E must be a type whose instances can be
compared to each other (e.g. they have a compareTo method).


}
     static <E extends Comparable<E>> void mergeSort(E []
array) {

}

}
```

Finally, write a Test class and check that your Sort class is capable of sorting chocolates, times and employees (from prev. task #4).

GOOD LUCK ☺   By Pakita Shamoi