Computer Science Division
Department of Mathematics
Faculty of Science

Level 3- Fall Semester
Course Code: Comp 301
Date: November 11, 2024

Sheet 6

Objective: upon successful completion of this sheet, students should be able to deal with abstract classes, interfaces, and special types of interfaces named "`Comparable`" and "`Comparator`".

1. Design the "`GeometricObject`" class that contains the following components.
   Data Members: `color` of type string and `filled` of type boolean.
   Methods: suitable constructor(s), setters, getters, and override the `toString()` method.
   Abstract Methods: `getArea()` and `getPerimeter()`.

   Design also the "`Circle`" class, which is a subclass of the `GeometricObject` class and contains the following components.
   Data Members: `radius` of type float.
   Methods: suitable constructor(s), setters, getters, and override the `toString()` method.

   Design also the "`Rectangle`" class, which is a subclass of the `GeometricObject` class and contains the following components.
   Data Members: `length` and `width` of type float.
   Methods: suitable constructor(s), setters, getters, and override the `toString()` method.

   Design also the "`Square`" class, which is a subclass of the Rectangle class and contains the following methods: suitable constructors, `setSide()`, `getSide()`, and override the `toString()` method.

Test the classes with an array of geometric objects, which contains different types of geometric objects (circles, rectangles, and squares). Then, show the information of geometric objects whose areas exceed `20`.

2. Design the "`Movable`" interface, which contains the following methods: `moveRight()` and `moveDown()` to increase the $x-$ and $y-$coordintes by a specified amount, respectively.
   `moveLeft()` and `moveUp()` to decrease the $x-$ and $y-$coordinates by a specified amount, respectively.

   Design the "`MovablePoint`" class that uses the above interface and contains the following components.
   Data members: the $x-$ and $y-$coordinates of type float, xAmount and yAmount of type float.
   Methods: suitable constructors and override the `toString()` method to return a string description of the movable point instance in the format (x, y) Amount= $(xAmount, yAmount)$.

   Test the class by declaring reference variables from the Movable interface and using them to call the methods of the `MovablePoint` class.

3. Design the "`ComparableSorting`" class with a static method , `sort()`, capable of sorting any array (i.e., generic) whose base type implements the Comparable interface.
   Modify the `GeometricObject` class to implement the comparable interface using the `getArea()` method to compare two geometric objects. Test the `sort()` method in the `ComparableSorting` class by applying it to an array of geometric objects.

4. Design the following set of classes for a banking system, determining the relationships between classes and whether they are abstract or concrete.

a. "`BankAccount`" class:

Data members: `balance` of type double.

Methods: suitable constructor, setter and getter for `balance`, `deposit()` to add a specified amount to the balance, `withdraw()` to subtract a specified amount from the balance, and `transfer()` to transfer a specified amount from the current bank account to another bank account.

b. "`SavingsAccount`" class:

Data members: `interestRate` of type double.

Methods: a suitable constructor, setter and getter for `interestRate`, `addInterst()` to add the earned interest to the account balance.

c. "CheckingAccount" class:

Data members: `transactionCount` of type integer to count the number of transactions in this account.

Static data members: `noOfFreeTransaction` of type integer and `transactionFees` of type double.

Methods: a suitable constructor, `transactionCount` should be updated through possible types of transactions (deposits, withdrawls, and transfers), and `deductFees()` to calculate the accumulated fees for all transactions, deduct them from the account balance, and reset the transaction count.

d. "`BankSystemApp`" class that contains the `main()` method to test the designed classes. Define a list of `BankAccount` objects, where some of them are declared as `SavingsAccount` objects, and the others as `CheckingAccount` objects. Perform deposit and withdrawal operations on some `SavingsAccount` objects and show their balances after interest calculation. Transfer some money from a `CheckingAccount` object to a `SavingsAccount` Object and to another `CheckingAccount` object. Show their balances after transfers and fee deduction. Finally, order all accounts in a descending order based on their balances using the `Comparator` interface.