Programming 1

Tutorial 6

Activity 1

Extend the below `BankAccount` example:

```
public class BankAccount {
    double balance;
    int transactions;
    public BankAccount(double initial) {
        this.balance = initial;
        this.transactions = 1;
    }

    public void deposit(double amount) {
        balance += amount;
        transactions++;
    }

    public void withdraw(double amount) {
        balance -= amount;
        transactions++;
    }

    public void monthlyFee() {
        this.withdraw(10);
    }
}
```

- Add a method named `annualInterest` to add annual interest to the account's balance.

  (*) Hint: *You should add an attribute called `interestRate` for this feature and modify the constructor to initialize this attribute.*

- A bank account should have the name of the account holder. Implement this feature.

  (*) Hint: *You should add an attribute called `holderName` for this feature and modify the constructor to initialize this attribute.*

- Add a method called `toString()` which returns a string of this format:
  (if the balance is negative, put the – sign before the dollar sign)

  `Benson, $117.25`

  `Mathew, -$17.50`

- Implement a method called `transfer` to send money from one bank account to another.

  There is a `$0.5` fee per transfer. Also, if the account does not have enough money to transfer and pay the fee, print out suitable error messages.
- Write a program called `BankAccountDemo` to test/demonstrate this `BankAccount` class.

Deliverable

BankAccount.java
BankAccountDemo.java

Activity 2

Implement a class `Car` with the following properties. A car has a certain fuel efficiency (measured in miles/gallon or liters/km—*pick one*) and a certain amount of fuel in the gas tank. The efficiency is specified in the constructor, and the initial fuel level is `0`. Supply a method `drive` that simulates driving the car for a certain distance, reducing the amount of gasoline in the fuel tank. Also supply methods `getGasInTank`, returning the current amount of gasoline in the fuel tank, and `addGas`, to add gasoline to the fuel tank.

Sample usage:

```
Car myHybrid = new Car(50); // efficiency of 50 miles per gallon
myHybrid.addGas(20); // Tank 20 gallons
myHybrid.drive(100); // Drive 100 miles
double gasLeft = myHybrid.getGasInTank(); // Get gas remaining in tank
```

You may assume that the `drive` method is never called with a distance that consumes more than the available gas. Create a `CarDemo` class with a *`main`* method that uses this `Car` class and show all of its features.

Deliverable

Car.java
CarDemo.java

Activity 3

(optional)

Write a program that asks the user for an integer and then prints out all its factors. For example, when the user enters `150`, the program should print:

```
2 3 5 5
```

(They are factors of 150 because 2 * 3 * 5 * 5 = 150)

Use a class `FactorGenerator` with a constructor `FactorGenerator(int numberToFactor)` and methods `nextFactor` and `hasMoreFactors`. Supply a class `FactorPrinter` whose `main` method reads a user input, constructs a `FactorGenerator` object, and prints the factors.

Hint

A prime number has only one factor, which is itself. A non-prime number has more than 1 factor, the factors are from 2 up, and smaller than the number. For example, consider the integer 150.

- Try to divide it by a factor of 2 until no longer divisible. We can divide it by 2 for 1 time to obtain 75. The first factor is 2.
- Use the next prime number, which is 3, as a factor. Divide 75 by 3 until no longer divisider, we can devide it for 1 time to obtain 25.
- The next prime is 5. This time, we can divide the number by 5 for 2 times.

So basically, we repeat this process:

1. Let factor = 2.
2. Divide the number by factor until not divisible.
3. If the number is 1, stop. Otherwise, use the next prime number as factor, then repeat from step 2.

We can see that, when the number has not been reduced to 1, there're still more factors to find. We can use this information to code the `hasMoreFactors()` method.

To find the next factor, we need to know the *last factor* and the *current value of the number*.

Deliverable

FactorGenerator.java

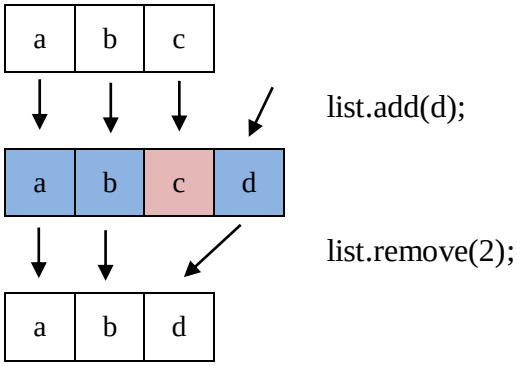
FactorPrinter.java

Activity 4

(optional)

As you know, arrays in Java have the limitation of fixed size. Sometimes we need arrays that have variable length. In order to do so, in this activity, you will create a type named `List` which behaves like a dynamic array of integers. This class should store the elements in an attribute

named `els`, which is an integer array. It should provide methods to `add`, `get` and `remove` elements.



- The `els` attribute should be initialized as an empty array (`length = 0`) in the constructor of this class.
- There should be an integer attribute named `size` to keep track of the list's length. It should be initialized as `0`.
- Method `add` is used to add a new value at the end of the list. It should be a `void` method which receives a single integer input parameter. In this method, you have to create an array which is one element bigger than the current `els` array. Transfer all elements from `els` to the new array. Then, assign the new value to last element of the new array. Finally, replace `els` with the new array.
- Method `remove` should perform a task which is opposite to that of the `add` method. It should also be a `void` method and receives one integer parameter. However, this input parameter is the index of the element to be removed. This method should create an array which is one element smaller than `els`. Then, transfer all values from `els`, except the element to be removed, over to the new array. Finally, replace `els` with the new array.
- Method `get` should return the value at a certain index.

Deliverable

List.java

ListDemo.java

Submission

Submit a **zip** file containing all Java programs to this tutorial's submission box in the course website on FIT Portal.