In Semester Examination-II, Winter 2018

IT622 Object Oriented Programming

Time: 90 Minutes | Max Points: 60

IMPORTANT NOTE: Answer all questions in question paper itself in provided space. Write answers neat and clean. Answers that are difficult to read may simply be ignored. May be you can first work in rough area and write final version as answer.

1. Given classes A, B, and C defined, you need to find output by each of method calls (labeled as S1 through S12) in main method below. If you cannot determine output due to some error, mention what type of error it is – syntax or run-time in your answer, and possible reason of error. Assume that error in a statement does not affect other statements; you can assume that error has been fixed.

[12]

```
public class A {
   public void f1(double x) { System.out.println("A.f1(): " + x ); }
   public void f2(String x) { System.out.println("A.f2(): " + x ); }
}
class B extends A {
   public void f1(String x) { System.out.println("B.f1(): " + x); }
   public void f2(String x) { System.out.println("B.f2(): " + x); }
}
class C extends B {
   public static void man( String[] args ) {
        A a = new C();
        a.f1(5.5);      //S1: __ C.f1(): 5.5_____________

        a.f2(5.5);      //S2: __Compilation Error: No such method_

        a.f1("Hello");  //S3: _Compilation Error: No such method_

        a.f2("Hello");  //S4: _ B.f2(): Hello ___________________

        B b = new C();
        b.f1(5.5);      //S5: __C.f1(): 5.5 ____________________

        b.f2(5.5);      //S6: _ Compilation Error: No such method _

        b.f1("Hello");  //S7: ___ B.f1(): Hello _________________

        b.f2("Hello");  //S8: ____ B.f2(): Hello _____________

        C c = new C();
        c.f1(5.5);      //S9: ____ C.f1(): 5.5_____________

        c.f2(5.5);      //S10: ____ C.f2(): 5.5 _____________

        c.f1("Hello");  //S11: ___ B.f1(): Hello ___________

        c.f2("Hello");  //S12:_____ B.f2(): Hello _____________

   }
   public void f1(double x) { System.out.println("C.f1(): " + x); }
   public void f2(double x) { System.out.println("C.f2(): " + x); }
}
```

2. What is output of following code? If error, what type of error (syntax/run-time) it is, and give possible reason of error. [5]

```
public class A {
    public A() { i++; j++; }
    public static void main(String[] args) {
        A[] a1 = new A[3];
        A a2 = new A();
        System.out.println("i=" + i + " j= " + j);
    }
    static int i = 0;
    int j = 0;
}
```

Error: j is not accessible in println of main

Consider **BankAcount** class discussed in lectures. Source code of the same is given here for your reference.

```
Public class BankAccount{
   public BankAccount()   {
       this.accno = ++ BankAccount.next_accno;
       this.balance = 0;
   }
   public BankAccount(double initialBalance)   {
       this.accno = ++BankAccount.next_accno;
       this.balance = initialBalance;
   }
   public void deposit(double amount)   {
       double newBalance = this.balance + amount;
       this.balance = newBalance;
   }
   public void withdraw(double amount) {
       double newBalance = this.balance – amount;
       this.balance = newBalance;
   }
   public long getAccno() { return accno; }
   public double getBalance() {return this.balance;}
   private final long accno;
   protected double balance;
   private static long next_accno=10000;
}
```

Suppose we need to extend the functionality of this class such that we are able to maintain history of transactions, i.e. withdraw and deposit performed on an account. To implement this, let us say we have identified a new class called **Transaction** – a *Transaction* object holds three values- Date, Transaction-Type, and Amount. Transaction-Type is a short value (0 or 1), where 0 for deposit and 1-for withdrawal. To store transaction history, let us say, an account object holds ordered collection of *Transaction* objects. In this exercise, you are required to do following –

3. What additional fields you will have in Bank Account class. Give proper declaration (only for additional fields)

[10]

Following in field declaration section

```
ArrayList<Transaction> transactions = new ArrayList<>();
```

OR

```
ArrayList<Transaction> transactions;
```

and initialization in constructor as

```
transactions = new ArrayList<>();
```

4. Provide implementation of **Transaction** class. Implement methods: (a) constructor that initializes required fields and accessor methods for required fields (let us say no mutator method is required).

[10]

```
public class Transaction {
    Transaction(Date dt, int trty, double amount ) {
        date = dt;
        tran_type = trty;
        this.amount = amount;
    }
    public Date getDate() {
        return date;
    }
    public int getTran_type() {
        return tran_type;
    }
    public double getAmount() {
        return amount;
    }
    Date date;
    int tran_type;
    double amount;
}
```

5. For added functionality, modify and rewrite following methods (you figure out what is to be coded in each method). You should be able to accommodate required implementation in provided space; still if you run out of space, can use extra sheet only for *findAverageBalance*.

[5+4+4+10]

(a) **constructor**, (b) **deposit,** (c) **withdraw**, and
(d) **findAverageBalance**: this method computes and returns weighted average balance over all transactions on the account. For example, in an account balance of 5500 remains for 10 days, 3400 for 25 days, and 6500 remains for 60 days, then the average balance = (5500*10+3400*25+6500*60)/(10+25+60) .

(a) Constructor:

```
    public BankAccount()   {
        this.accno = ++BankAccount.next_accno;
        this.balance = 0;
        transactions = new ArrayList<>();
    }

    public BankAccount(double initialBal)   {
        this.accno = ++BankAccount.next_accno;
        this.balance = initialBal;
        transactions = new ArrayList<>();
        transactions.add(new Transaction(new Date(), 0, initialBal));
    }
```

(b) Deposit

```
    public void deposit(double amount)   {
        double newBalance = this.balance + amount;
        this.balance = newBalance;
        transactions.add( new Transaction(new Date(), 0, amount));
    }
```

(c) Withdraw

```
    public void withdraw(double amount) {
        double newBalance = this.balance - amount;
        this.balance = newBalance;
        transactions.add( new Transaction(new Date(), 1, amount));
    }
```

(d) findAverageBalance:

```
    public double findAverageBalance() {
        double avg = 0;
        double amount = balance;
        Date date = new Date();
        int n = transactions.size();
        int n_days = 0;
        double sum_amount = 0;
        while ( n > 0){
            Transaction t = transactions.get(n-1);
            Date tdt = t.getDate();
            long time = date.getTime() - tdt.getTime();
            long days = time / (24 * 60 * 60 * 1000);
            sum_amount += amount * days;
            n_days += days;
            if (t.getTran_type() == DEPOSIT)
                amount -= t.getAmount();
            else
                amount += t.getAmount();
            date = tdt;
            n--;
        }
        avg = sum_amount / n_days;
        return avg;
    }
```