UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2002

BEng Honours Degree in Computing Part I
MEng Honours Degrees in Computing Part I
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute

PAPER C123

PROGRAMMING II

Wednesday 1 May 2002, 14:00
Duration: 90 minutes
(Reading time 5 minutes)

Answer THREE questions

Paper contains 4 questions
Calculators not required

Section A *(Use a separate answer book for this Section)*

1a i) Describe what the Abstract Data Type (ADT) *List* is, and give the *List*'s access procedure headers with post-conditions, also specifying the exception cases.

ii) Consider the ADT *Stack*. Give its access procedures' headers with the post-conditions, also specifying the exception cases.

iii) Give the axioms that any implementation of the ADT *Stack* must satisfy.

iv) Explain the difference between the ADTs *List* and *Stack*.

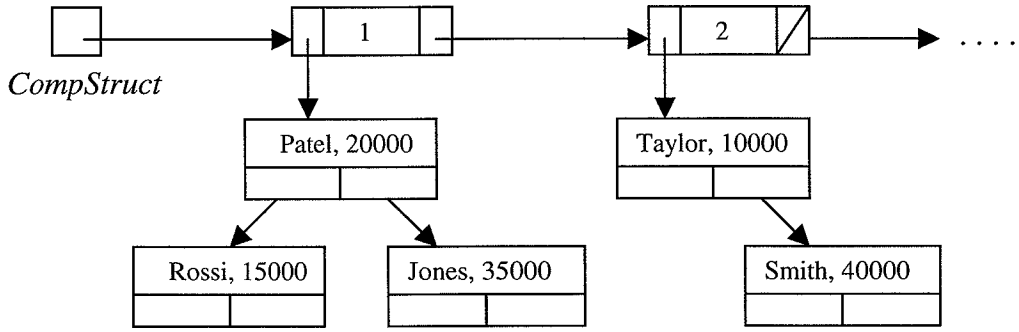
b Assume the availability of an ADT *List* with its own access procedures, and the availability of an ADT *Stack* with its own access procedures. Assume the order of access in a list to be from the first to the last item in the list. Write the Java code for the following two high-level access procedures:

i) *Stack* ListtoStack(*List* L)
//pre: L is a list
//post: returns a stack containing the items of L, with the access order identical to
//that of L; L is left unchanged

ii) *List* ReverseStacktoList(*Stack* S)
//pre: S is a stack
//post: returns a list containing the items of S, where the access order of L is the
//reverse of the access order of S.

The two parts carry, respectively, 60% and 40% of the marks.

2 The internal organization (*CompStruct*) of a company is composed of ten different sectors, uniquely numbered from 1 to 10. People employed within the same sector are organized into a tree structure, sorted on the basis of their salaries. Within the same sector, employees all have different salaries. For each employee the organization includes his/her surname and salary. The diagram below gives a partial example of such an organization. In sector 1 the employees are Rossi with the lower salary 15000, Jones with salary 20000 and Patel with higher salary 35000. In sector 2, the employees are Taylor with salary 10000 and Smith with higher salary 40000.



a. Assume *CompStruct* to be the definition of an ADT. Give the Java type declarations for this ADT *CompStruct* in full, including the type declarations of all the data structures needed to implement it. Do not encapsulate the declaration of an employee into a separate type.

b. Assume now the existence of a standard Binary Search Tree ADT. Give the Java type declaration for the ADT *CompStruct* using this standard ADT.

c. Write the Java code for the *CompStruct*'s access procedure *EmployeeName* specified below. To do so, use the Java type declaration you have given to part (b) and assume the existence of the Binary Search Tree's access procedure String retrieve(int sal). This procedure returns the name of the employee in a given tree, whose salary is equal to sal.

String EmployeeName(int sectnum, int sal)
//pre: *sectnum* is a sector number, and *sal* is the salary of an employee in this sector.
//post: returns the surname of the employee in the given *sectnum*, whose salary is *sal.*

Give also the implementation of any auxiliary method or other access procedure of *CompStruct* that might be needed.

The three parts carry, respectively, 30%, 20%, 50% of the marks.

Section B *(Use a separate answer book for this Section)*

3a Implement a class *CarRentalCustomer*. Class *CarRentalCustomer* inherits class *Customer* and has the following fields/attributes:

(i) "renting", states whether "this" customer is currently renting a car;

(ii) "rentedCars", an array of type RentalCar for this customer;

(iii) "surname", the customer's surname (inherited from Customer);

(iv) "firstName", the customer's first name (inherited from Customer);

(v) "id", a numerical customer id;

(vi) "rentalTime", an array of the number of rental days for each rented car; and

(vii) "creditCardDetails", a primitive type containing details of the customer's credit card(s).

b Write a suitable constructor method for class *CarRentalCustomer*; it initializes ALL fields and has a first name, surname, and list of type int -- the array for rental times -- as parameters; however, first name and surname are to be initialized by class Customer; you may assume that customers rent no more than 20 cars at a time;

c Write methods for *CarRentalCustomer*:

(i) "status", an abstract method that returns a description of the customer's importance to the rental company, such as `platinum', `advantage gold', etc;

(ii) "deleteCar", an abstract method that deletes a specific RentalCar -- its only parameter -- from rentedCars, if possible;

(iii) "addCar", an abstract method that adds a specified RentalCar to rentedCars, if possible;

(iv) "getBillingInformation", returns the customer's credit card details.

Finally: Where would you put these methods within class CarRentalCustomer? What changes, if any, do you have to make to the code written in the previous part?

The three parts carry, respectively, 25%, 50% and 25% of the marks.

4a Explain the crucial consequences of

(i) declaring a primitive type (e.g. int) as final;

(ii) declaring a class as final;

(iii) storing a class in folder F and file C;

(iv) declaring a field to be private;

(v) declaring a field to be *protected.*

b Write a Java method *LastEvenIfAny* that has as parameter an integer array and returns the rightmost even entry in that array, provided it exists. Otherwise, it returns -1. For example [12, 4, 3, 8, 9, -2, 5] returns -2 and [3, -7, 11] returns -1. Make sure that your code returns -1 if the array is "empty".

c Consider the following Java source code:

```
public interface ITransaction {
      public double calcTotal();
}

public class RentalTransaction implements ITransaction {
      private double amount;
      public void RentalTransaction(int price) {
          amount = price;
      }
      public int calcTotal(double discount) {
          return amount - discount;
      }
}

public class AuctionTransaction {
      private double amount;
      public AuctionTransaction(double price) {
          amount = (int)price;
      }
      public abstract int calcTotal(double heftyFee) {};
}

public class TransactionRunner {
      public static void main(String[] args) {
          RentalTransaction booking = new RentalTransaction(586);
          ITransaction iBooking = (ITransaction)booking;
          if (iBooking instanceof RentalTransaction) {
                double myTotal = iBooking.calcTotal(1.0);
          }
          AuctionTransaction auction = new AuctionTransaction(12000000.35);
          double myBill = auction.calcTotal();
      }
}
```

Identify eight errors in this source code that the compiler will detect and complain about. Explain the nature of these errors in each instance that you identify.

The three parts carry, respectively, 25%, 25% and 50% of the marks.