**Assignment : Object, Class, and Encapsulation**

**1**.   Create a class called **Author** is designed as follows: It contains:

•     Three  private  instance  variables:    name  (String), email (String), and gender (char of either ‘m’ or ‘f’).

•   One constructor to initialize the name, email and gender

with the given values.

And, a class called **Book** is designed as follows: It contains:

•   Four private instance variables: name (String), author (of the class Author you have just created), price (double), and qtyInStock (int).

Assuming that each book is written by one author.

•     One  constructor  which  constructs  an instance             with    the

values given.

•    Getters and setters: getName(), getAuthor(), getPrice(),

setPrice(), getQtyInStock(), setQtyInStock(). There is no setter for name and author.

Write  the  class  BookInfo  (which  uses         the  “Author”           class written earlier).

**Try:**

1.  Printing  the  book  name,  price and        qtyinstock from BookInfo object.

(Hint: aBook.getName())

2.  After  obtaining  the  “Author”  object, print  the          Author

(name, email & gender) of the book.

**2**.   The machine has two main components:

A built-in cash register and several dispensers to hold and release the products.

Define   class   cashRegister   in   Java            with    the                following descriptions :

Private Members: cashOnHand of type integer

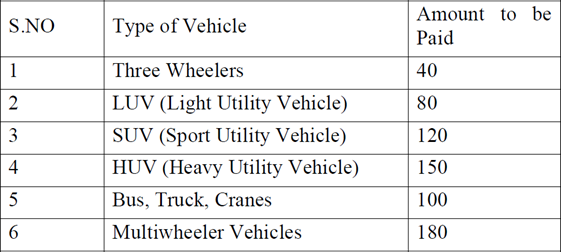
Public Members: A default constructor cashRegister() sets the cash in the register to 100.

A constructor cashRegister(int) sets the cash in the register to a specific amount.

A function getCurrentBalance() which returns value of cashOnHand

A function acceptAmount(int) to receive the amount deposited by the customer and update the amount in the register.

**3.**

  
Assume you see a tollbooth at a highway. It is rule that all vehicles passing by the booth are expected to pay the following amount:

* In few cases, at times some vehicle goes by without paying toll. The tollbooth keeps track of the number of vehicle that have passed by, and of the total amount of money collected. Implement this tollbooth with a class called tollbooth. The two data items are a type unsigned int array to hold the number of Vehicles, and a type double to hold the total amount of money collected.A constructor initializes both of these to 0.A member function called payingVehicle() increments the car total and adds 0.50 to the cash total. Another function, called nopayVehicle(), increments the Vehicle total but adds nothing to the cash total. Finally, a member function called display() displays the two totals.
* Include a program to test this class. This program should allow the user to push one key to count a paying Vehicle, and another to count a non-paying Vehicle.

Develop java application must print out the total Vehicles and total cash and then exit.

**4**.  Declare a class **Employee** having empId, empName, empOrganization as members. Salaried employees are paid a fixed weekly salary regardless of the number of hours worked and hourly employees are paid by the hour and receive overtime pay for all hours worked in excess of 40 hours.

* Design your class such that it has appropriate members such as designation, experience, get and set weekly salary, weekly\_gross\_salary\_calculation based on number of hours worked and so on.
* Define appropriate constructors, the Main method to store array of ‘N’ salaried employees and hourly employees accessing their pay details, A method to find and display details of hourly employees who are drawing salary more than 20,000/per month.

Provide Java code for this scenario.

**5**.  Define a **Library class** which has drawBook(), returnBook(),checkStaus() and reserveBook() methods and necessary attributes.

All the methods are tagged with public in the following ways.

(a)    Using drawBook ()-get the required book based on the title.

(b)   Using checkStatus()-user book returned date details.

(c)   Using reserveBook()-block or reserve particular book for their account.

Using returnBook()-Return the drawn books. Implement this scenario and create user friendly Library management applications.

**Topic : Inheritance, Polymorphism, Abstract Class, and Interface**

**1.**  Create a class named 'Employee' having the following fields:

i.            Name, Age, Phone number, Address, Salary

ii.           It also has a method named 'printSalary' which prints the salary of the members.

iii.         Two classes 'Permanent' and 'Contract' inherits the 'Employee' class. The 'Permanent' and 'Contract' classes have data members 'Experience, Service\_record ' and 'Period, Allowances' respectively. Now, assign name, age, phone number, address and salary to a Permanent and a Contract employee by making an object of both of these classes and print the same.

**2.**  Implement the following inheritance hierarchy.

i.        Create a class named BasketballPlayer with the attributes of name(String), team(String), height(int), weight(int), speed(int), position(String) as protected members.

ii.        Create 3 types of constructors to initialize the data members.

a.                     No argument constructor which assigns 0 for int data members and Null for String data members

b.                    3 string argument constructor which assigns strings for string variables and 0 for int data members

c.                       Third constructor with 6 arguments which assigns values to all the data members using received values.

iii.        Also create accessors for all instance data and getValue method will test various instance data and return an int using nested if-else logic based on the table below. All methods should be public.

|  |  |  |
| --- | --- | --- |
| Position | other Conditions | Value |
| Center | height>=80, weight between 220 and 250 | 10 |
| Center | height>=78, weight between 210 and 260 | 5 |
| Center | height>=76, weight between 180  and 220 | 3 |
| Center |  | 0 |
| Forward | height>=80, Speed>7 | 10 |
| Forward | height>=78, Speed>5 | 5 |
| Forward | height>=76, Speed>4 | 3 |
| Forward |  | 0 |
| Guard | height>=78, Speed>7 | 10 |
| Guard | height>=76, Speed>6 | 5 |
| Guard | height>=74, Speed>5 | 3 |
| Guard |  | 0 |

iv.        Create a class named CollegeBasketballPlayer which extends BasketballPlayer with data members eligibilityRem(int) and role (String) as private or protected members. Have 3 different constructors, a no-argument constructor, 3 argument constructor(name,position,team) and a constructor receives all the parameters. Use super(); for no argument constructor and  super(…) for other constructors. By default assign eligiblityRem=4, role=”bench” for no and 3 argument constructors. Assign proper parameter values in the 3rd constructor. Add accessors for these two data. Finally, add a method called draftable which returns a boolean based on the following logic:

 A player is draftable if he/she is a “starter” (role) whose value (as obtained by super.getValue()) > 4 or is a “bench” player (role) whose value >= 8.

v.            Implement ProBasketballPlayer by extending BasketballPlayer. Add

2 data members: yearsInLeague(int) and role (String). Add 3 constructors similar to what you had for CollegeBasketballPlayer. For the first two constructors, yearsInLeague should be initialized to

0  and  role  to  “bench”.  Finally,  implement  a  method  called

newContractValue which returns an int value based on the logic in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| Value | Years             in League | Role | Return Value |
| >=8 | >=10 | Starter | $12,00,000 |
| >=5 | >=8 | Starter | $10,00,000 |
| >=8 | >=8 | Starter | $8,00,000 |
| >=8 | >=10 | Bench | $8,00,000 |
| >=5 | >=8 | Bench | $5,00,000 |
| >=8 | >=8 | Bench | $2,00,000 |

 vi.        Finally create a user class with a main method and implement the above given hierarchy.

**3.**  Create an abstract class 'Account' with data members amount(double), InterestRate(double) and an abstract method ‘calculateInterest(). Create the following subclasses

**FDAccount**

double interestRate double amount

int noOfDays

int ageOfACHolder

abstract double calculateInterest()

**SBAccount**

double interestRate double amount

abstract double calculateInterest()

**RDAccount** double interestRate double amount

int noOfMonths;

double monthlyAmount;

abstract double calculateInterest()

Calculate the interest based on the account type and display the same.

**4.**  Implement the above said example using interface concept.

**5.**  All vehicles need servicing from time to time. Suppose we want to create two classes named Car and Bike having a method that returns the time after which servicing is required for the vehicle. Here, we can create an interface named Servicing having an abstract methods named getServiceTime(), displayServiceHistory() and can make the classes Car and Bike implement the interface Servicing and thus implement its methods.