Lab Guide for Object Oriented Programming Using Java

Education and Research

Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

CONFIDENTIAL

Author(s)	Meenakshi S
Authorized by	Mr. Srikantan Moorthy
Creation/Revision Date	Feb-2010
Version	2.2

COPYRIGHT NOTICE

All ideas and information contained in this document are the intellectual property of Education and Research Department, Infosys Technologies Limited. This document is not for general distribution and is meant for use only for the person they are specifically issued to. This document shall not be loaned to anyone, within or outside Infosys, including its customers. Copying or unauthorized distribution of this document, in any form or means including electronic, mechanical, photocopying or otherwise is illegal.

Education and Research Department
Infosys Technologies Limited
Electronic City
Hosur Road
Bangalore - 561 229, India.

Tel: 91 80 852 0261-270
Fax: 91 80 852 0362
www.infosys.com
mailto:E&R@infosys.com

CONFIDENTIAL

Document Revision History

Version	Date	Author(s)	Reviewer(s)	Comments
1.0	Mar-2009	Vani Vasudevan Lesitha Mary James	Meenakshi S Nishit Goel	Restructure 2009
1.1	Jun-2009	S.Meenakshi Lesitha Mary James	Nishit Goel	Changes made as per feedback from pilot batch rollout
1.2	Aug-2009	S.Meenakshi Lesitha Mary James	Nishit Goel	Modified assignments based on case study
2.0	Feb-2010	Meenakshi S Lesitha Mary James Anooja Mary Jacob	Lakshmi D L Nishit Goel	Changes made to incorporate the threaded assignments and feedbacks
2.1	May-2010	Anooja Mary Jacob	Meenakshi S	Minor changes made after the pilot
2.2	Jun-2010	Anooja Mary Jacob Lesitha Mary James	Meenakshi S	Minor changes made after the pilot

CONFIDENTIAL

Contents

COPYRIGHT NOTICE II

DOCUMENT REVISION HISTORY I

CONTENTS 1

CONTEXT FOR DAY 1 3

DAY 1 ASSIGNMENTS 3

ASSIGNMENT 1: CASE STUDY 3
ASSIGNMENT 2: IDENTIFY THE OO CONCEPTS 4
ASSIGNMENT 3: CLASS DIAGRAM 5
ASSIGNMENT 4: OBJECT ORIENTED CONCEPTS 6
ASSIGNMENT 5: OBJECT ORIENTED TERMINOLOGIES 7
ASSIGNMENT 6 : CASE STUDY OF SPEDFAST COURIER COMPANY 8
ASSIGNMENT 7: CASE STUDY OF D-RAIL INC. 9

CONTEXT FOR DAY 2 10

DAY 2 ASSIGNMENTS 10

ASSIGNMENT 1: CLASSES AND OBJECTS 10
ASSIGNMENT 2: CLASSES AND OBJECTS 12
ASSIGNMENT 3: UNDERSTANDING JAVA ARCHITECTURE 13
ASSIGNMENT 4: CONTROL STRUCTURES AND OPERATORS 14
ASSIGNMENT 5: CONTROL STRUCTURES 15
ASSIGNMENT 6:TYPE CASTING 17
ASSIGNMENT 7: DEBUGGING - TYPE CASTING 17
ASSIGNMENT 8: LOOPS 18
ASSIGNMENT 9: DEBUGGING - THIS REFERENCE 19
SELF REVIEW 20

AT THE END OF DAY 2….. 23

AT THE END OF DAY 2 YOU HAVE THE FOLLOWING FILES: 23

CONTEXT FOR DAY 3 24

DAY 3 ASSIGNMENTS 24

ASSIGNMENT 1: ARRAYS 24
ASSIGNMENT 2: MULTI DIMENSIONAL ARRAYS 25
ASSIGNMENT 3: STRINGS 26
ASSIGNMENT 4A: DEFAULT CONSTRUCTOR 28
ASSIGNMENT 4B: DEFAULT CONSTRUCTOR AND STATIC 29
ASSIGNMENT 5: STRINGS 32
ASSIGNMENT 6: COMMAND LINE ARGUMENTS 33
ASSIGNMENT 7: MULTIDIMENSIONAL ARRAY(JAGGED ARRAY) 34
ASSIGNMENT 8: ARRAY OF STRINGS 34

CONFIDENTIAL

SELF REVIEW 36

AT THE END OF DAY 3….. 39

AT THE END OF DAY 3 YOU HAVE THE FOLLOWING FILES 39

CONTEXT FOR DAY 4 40

DAY 4 ASSIGNMENTS 40

ASSIGNMENT 1: PARAMETERIZED CONSTRUCTOR AND CONSTRUCTOR OVERLOADING 40
ASSIGNMENT 2: METHOD OVERLOADING 42
ASSIGNMENT 3: RELATIONSHIPS – 'HAS-A' RELATIONSHIP 43
ASSIGNMENT 4: RELATIONSHIPS – 'IS A' RELATIONSHIP 45
ASSIGNMENT 5: RELATIONSHIPS – USES A RELATIONSHIP 49
ASSIGNMENT 6: METHOD OVERRIDING 50
ASSIGNMENT 7: DYNAMIC POLYMORPHISM 52
SELF REVIEW 57

AT THE END OF DAY 4….. 61

AT THE END OF DAY 4 YOU HAVE THE FOLLOWING FILES 62

CONTEXT FOR DAY 5 62

DAY 5 ASSIGNMENTS 62

ASSIGNMENT 1: ABSTRACT CLASS AND METHOD 62
ASSIGNMENT 2: INTERFACES 64
ASSIGNMENT 3: COURSE REGISTRATION – THE COMPLETE PROGRAM 65
ASSIGNMENT 4: PACKAGES 69
ASSIGNMENT 5: PACKAGES FOR COURSE REGISTRATION SYSTEM 70
ASSIGNMENT 6: COURSE REGISTRATION SYSTEM - OBSERVATIONS 71
SELF REVIEW 71

AT THE END OF DAY 5….. 76

AT THE END OF DAY 5 YOU HAVE THE FOLLOWING FILES 77

CONFIDENTIAL

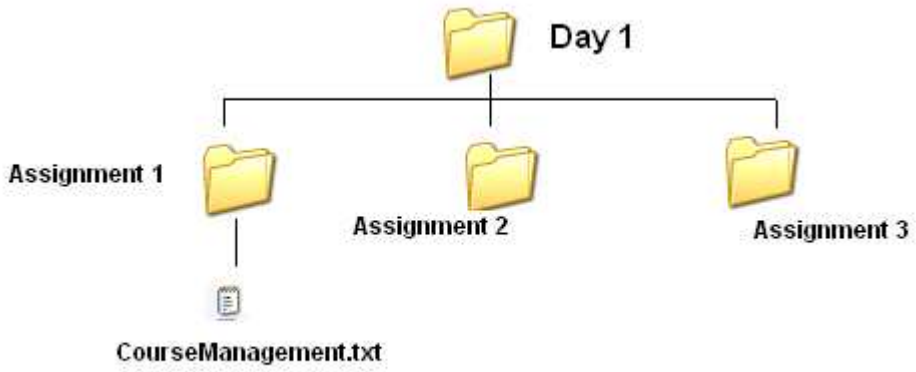
Context for Day 1

The following are the topics and concepts covered in the assignments as part of the Day 1 assignments

- Object Oriented Concepts
- Object Oriented Analysis and Design

Day 1 Assignments

Instructions:
Create a folder for Day 1 assignments in your home folder
Create a folder for each assignment and make the respective files related to that assignment in that folder as shown below:



Assignment 1: Case Study

Objective: Analysis of the following Case Study from the perspective of Object Oriented Approach

Problem Description: Course Registration System

A Course Registration System needs to be developed for an engineering college. The college wants an automated system to replace its manual system for the purpose of registration of students to branches and calculation of fees for each year. The engineering college provides graduation courses in various branches of engineering.

The system will be used by the admin staff to register students admitted to the college to the branches at the time of joining the college and also to calculate the yearly fees for the students. The student has to register every year for the next academic year. The Admin takes care of the yearly registration of the students and the calculation of yearly fees. The system needs to be authenticated with a login id and password.

Registration of a student to a branch is based on the qualifying exam marks and the entrance counseling. For every branch, a yearly branch fee is applicable. Discounts are given to the branch fees of the first year based on the qualifying exam marks. There is a registration fees also applicable to the first year students. Students can opt to be a day scholar or hostelite. Yearly bus fees is applicable for all the day scholars based on the distance of travel. Yearly hostel fees is applicable for all the hostelites. Yearly infrastructure fees and library fees are also applicable to all the students. Admin calculates the yearly college fees for each student and the college fees include all the fees specified earlier based on the type of student. Admin will provide a printed receipt of the fees to the students once the annual college fees have been paid.

At the time of registration, student has to provide the permanent address and in case the student is opting to be a day scholar, he/she has to provide the residential address also.

Assumption:

1. Decision of to which branch a student is allocated, is not within the scope of this case study

Answer the following questions:

1. Identify all the actors of the Course Registration System

2. Identify the activities i.e use cases of the system

3. Connect the actors to the use cases and draw the use case diagram for the system

Note:

- The use case diagram should be drawn in a Microsoft Word Document and submitted as part of the assignments

Estimated time: 40 mins

Summary of this assignment:

In this assignment, you have learnt

- How to analyze and design the Use case diagram for a given problem statement

Assignment 2: Identify the OO concepts

CONFIDENTIAL

Objective: Analysis of the scenarios picked from the case study discussed in Assignment 1 of Day 1 and identification of the OO concepts associated to the scenarios

Problem Description: Identify the OO concepts associated with the below given scenarios

1. There are around 10000 students studying in various branches of engineering in the college. Every student has some attributes and performs some activities. How can the attributes and activities be represented using OO concepts .

2. Students can opt to be a Day Scholar or a Hostelite. In case of Day Scholar, bus fees is applicable based on the distance from the residential location to the college. In case of Hostelite, hostel fees is applicable. Identify the OO concept discussed .Identify the generalized and specialized classes for this scenario.

3. The Admin staff calculates the fees to be paid for all students irrespective of the type of student. The formula for calculation however depends on the kind of student. Which is the OO concept discussed here?

4. Certain attributes of the student need to be accessed by the admin staff directly, and certain attributes need not be accessed. How can this be implemented and what is the OO concept discussed here?

Estimated time: 15 mins

Summary of this assignment:

In this assignment, you have learnt

- How to analyze a scenario and apply the OO concepts

Assignment 3: Class Diagram

Objective: Understand the scenario and draw the class diagram

Problem Description: There are around 10000 students studying in various branches of engineering in the college. Admin allocates a unique id for each student at the time of joining the college. The student has to fill a joining form specifying the following: student name, branch name, qualifying exam marks, permanent address and residential status. Residential status can be either 'D' (Day Scholar) or 'H' (Hostelite). In case, student is opting to be a Day Scholar, residential address and distance to the college also has to be filled. In case, the student is opting to be a Hostelite, the hostel name, room type and room number also has to be filled.

Answer the following questions:

CONFIDENTIAL

1. Identify the classes
2. Identify the generalized and specialized classes
3. Identify the attributes/activities associated to the generalized class
4. Identify the attributes/activities associated to the specialized classes
5. Draw the class diagram and mark the class name, attributes and activities of each class

Note:

- The class diagram should be drawn in a Microsoft Word Document and submitted as part of the assignments

Estimated time: 45 mins

Summary of this assignment:

In this assignment, you have learnt

- How to analyze a scenario and identify classes, attributes and activities related to a class
- How to draw a class diagram

Assignment 4: Object Oriented Concepts

Objective: Understand the various Object Oriented concepts and their definitions

Problem Description: Fill in the blanks for the below given statements

1. The properties exhibited by a complex system are ____, _____, _____ and ____
2. A _____ is a software design that describes the common attributes and behavior of objects
3. OO concept derived from the words meaning "In a Capsule" is _____
4. A class diagram is represented by a _____ with the components of a class
5. _____ are used to expose or hide the attributes/activities of a class
6. UML diagram which can be used to identify the functionalities of the system under development is called _____
7. OO concept wherein a class shares some common structure or behavior with one or more classes is called _____

CONFIDENTIAL

8. OO concept which refers to an object's ability to behave differently depending on its type is called _____

9. The Unified Modelling Language (UML) is a language for specifying, constructing, _____, and _____ the software system and its components

10. Process of focusing on essential details and ignoring non-essential details is called _____

Estimated time: 20 mins

Summary of this assignment:

In this assignment, you have learnt

- Object Oriented concepts and their definitions

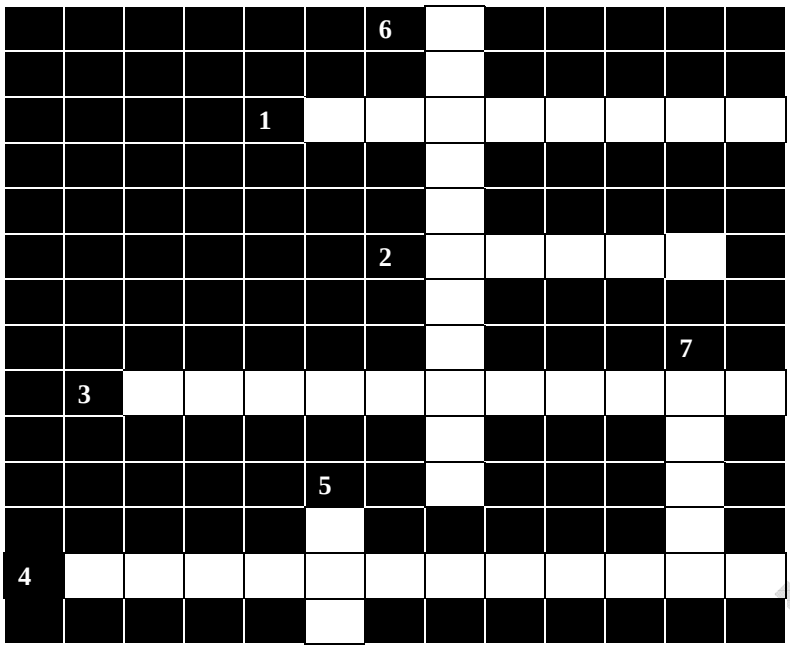
Assignment 5: Object Oriented Terminologies

Objective: Understand the various Object Oriented Terminologies

Problem Description: Fill in the crossword based on the scenario and clues given below:
Jane and Johan are two students who have joined to the computer science branch of the engineering college. Jane has opted to be a day scholar and Johan has opted to be a hostelite. The admin staff who is taking care of the Computer Science branch is Jack. Jack registers both the students to the first year as well as calculates their fees for the first year. The fees includes registration fees, branch fees, bus fees, infrastructure fees and library fees for Jane. For Johan, it includes registration fees, branch fees, hostel fees, infrastructure fees and library fees. Once the students pay the fees, the receipt is handed over to them by Jack.

1.	Jane is an _____ of Student class
2.	Jack who is the admin staff represents an _____ of the course registration system
3.	Jane and Johan are two kinds of students. This represents the _____ OO concept.
4.	The Admin staff calculates the fees to be paid for all students irrespective of the type of student. The formula for calculation however depends on the kind of student. This represents the _____ OO concept.
5.	The notation that can be used for representing the course registration system
6.	Jack has access to some student details which are essential for him. This represents the _____ OO concept
7.	The attributes and activities of a student can be bundled into a _____

CONFIDENTIAL



Estimated time: 20 mins

Summary of this assignment:

In this assignment, you have learnt

- Object Oriented Terminologies

Assignment 6 : Case study of SpedFast Courier Company

A courier company "SpedFast" wants to automate their process of collecting and dispatching shipments. The company has a set of customers, many of whom are already in the company's database. Each customer has a unique Customer ID. Each customer has an address as well, which is used as "Sender's address" for shipments.

Customers approach the *Dispatchers*[1] at SpedFast counters to send shipments. One customer can send as many shipments as required. The Dispatcher initiates a shipment after the payment from the customer.

Each shipment has a Shipment ID associated with it. A shipment also has a Recipient's address, priority, weight of the shipment, shipping date and date of delivery. The amount is calculated based on the weight of the shipment and priority. Currently SpedFast provides "LOW, NORMAL and URGENT" priorities with different rates on shipments.

[1] **Dispatcher:** The person who dispatches shipments in a postal organization or courier company.

CONFIDENTIAL

SpedFast also intends to have a Greeting Shipment service for customers to send Gifts for celebrations. This is similar to the normal shipments, but it also carries a "Greeting message" from the customer.

Step 1: Design the Use Case diagram

Estimated time: 10 mins

Summary of this assignment:

In this assignment, you have learnt

- To understand the requirements and design the Use Case diagram

Assignment 7: Case study of D-Rail Inc.

D-Rail Inc. is a metropolitan railway corporation operating in a large city and its suburbs. D-Rail has an ambitious plan to computerize its ticketing system. The IT department of D-Rail has come up with an OO design for their ticketing system.

The ticketing system addresses the needs of the metro rail service of D-Rail between the city and its suburbs.

Requirements of the Ticketing System:

A Ticketing Clerk at one of the D-Rail counters can issue the following kinds of tickets to commuters:

One Time ticket: Used for one way travel. It has to be used the same day it is issued.

Return ticket: Used for two way travel. Onward journey has to be done on the date of issue. However, the return journey can be done any time within a week from the date of issue.

Multi-use ticket: Used for traveling many times between an origin and destination stations (onward or return). Multi-use tickets are available 10, 15 and 20 trips denominations, which are also specified in the ticket. However the ticket has an expiry date which is two months from the date of issue. Every time the ticket is used, one trip is deducted.

All tickets have information on the origin and destination stations.

All the tickets issued are magnetic tickets which store the entire ticket information. This information will be verified by the ticket reading machine at the entry points.

Step 1: Design the Use Case diagram

CONFIDENTIAL

Step 2: Identify the classes

Estimated time: 15 mins

Summary of this assignment:

In this assignment, you have learnt

- To understand the requirements and design the Use Case diagram and identify classes

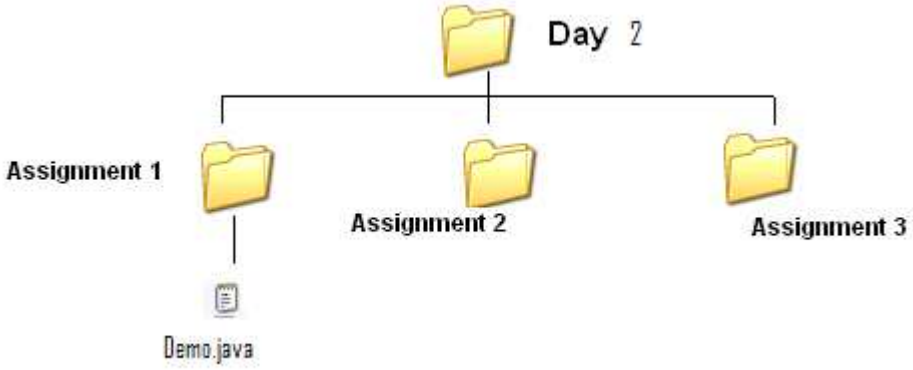
Context for Day 2

The following are the topics and concepts covered as part of the Day 2 assignments

- Classes and Objects
- Operators
- Type casting
- Control Structures
- Debugging and Logic

Day 2 Assignments

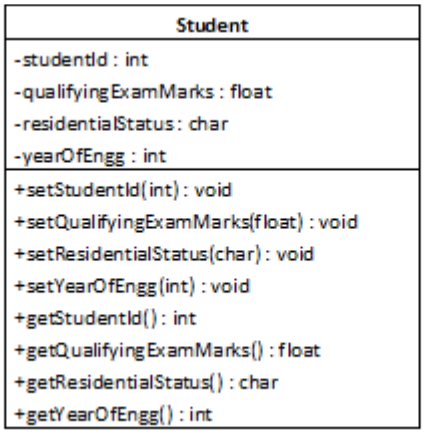
Instructions:
Create a folder for Day 2 assignments in your home folder
Create a folder for each assignment and make the respective files related to that assignment in that folder as shown below:



Assignment 1: Classes and Objects

Objective: Creation of a Student Class and instantiation of objects

Problem Description: On Day 1, the classes for the course registration system have been identified. The Student class with the attributes and methods is to be implemented in Java as per the class diagram given below:



Step 1: Create a file called **Demo.java** using TextPad

Step 2: Define the class ***Student*** as per the class diagram specified

Step 3: Define all the setter and getter methods of Student Class

Step 4: Create a class **Demo** with the main method

a. Create a reference variable of Student class with the name **student** and instantiate the same
b. Invoke the corresponding setter methods to set the values for the instance variables as follows:

studentId	1001
qualifyingExamMarks	95.0f
residentialStatus	D
yearOfEngg	2

c. Using the corresponding getter method, display the details as follows:

Student Id :_____
Qualifying Marks :_____
Residential Status :_____
Current Year of Engineering :_____

Step 5: Compile the program (Refer the document Compilation and Execution of a Java Program.pdf supplied along with the lab guide)

Estimated time: 25 minutes

Summary of this assignment: In this assignment, you have learnt

- To create a class , instantiate objects , invoke methods using an object
- To compile and execute a Java program

Assignment 2: Classes and objects

Objective: Debug a program containing classes and objects and understand the common programming errors

Problem Description: Consider the revised Demo class with the main method given below. Identify the errors and debug the same to get the output as shown:

```
class Demo{
      public static void main(String args[]){
            Student student;
            student.setStudentId(1002);
            student.setQualifyingExamMarks(68);
            student.setResidentialStatus('H');
            System.out.println("Student Id:"+student.studentId);
            System.out.println("Qualifying marks:"+getQualifyingExamMarks());
            System.out.println("Residential Status:"+getResidentialStatus());
            System.out.println("Year Of Engineering:"+getYearOfEngg());
      }
}
```

Output expected:

```
Student Id:1002
Qualifying Marks:68.0
Residential Status:H
Year Of Engineering:3
```

Note:

- Initialization of variables before usage is important
- Invocation of methods is done with the help of objects
- Private access specifier is accessible only within the class where it is declared

Estimated time: 20 minutes

Summary of this assignment:

In this assignment, you have learnt

- Debugging the common errors while creating objects

Assignment 3: Understanding Java architecture

Objective: Understand the creation of byte codes and .class files

Problem Description: Java programs are compiled to get the byte codes, which are verified and executed by the JVM. Consider the Java program created in Assignment 1 of Day 2 to explore and understand byte codes.
Perform the following steps:

Step 1: Go to the path where the Demo.java file is stored and answer the following questions:

a) How many class files do you see?
b) How many source files (.java) do you see?
c) Try opening the .class file using TextPad, What do you observe**?**

Step 2: Write the verification process for byte code

a) Open the .class file and make the modification (add a space after the last line) and save it.
Note: The .class file can be opened in TextPad, the content of .class file is not in readable format
b) Without compiling execute the same program. What do you observe?
c) Compile and execute the program what is difference you could observe?

Step 3: Remove the Student class written in Assignment 1 of Day 2 from **Demo.java** file and place it into another file called **Student.java**. Compile Student.java and Demo.java (containing the Demo class with the main method) separately and Execute Demo.java. Repeat Step 1 and answer the questions. Is there any difference in the answers?

CONFIDENTIAL

Estimated time: 15 minutes

Summary of this assignment:
In this assignment, you have learnt
- Java architecture

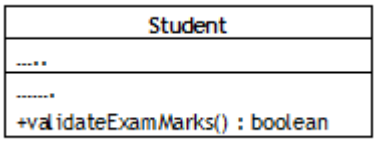
Assignment 4: Control structures and operators

Objective: Write programs to understand Control statements, Operators and boolean data type

Problem Description: The validation of the examination marks is to be done. The range of exam marks is between 65 and 100 (both inclusive). This is to be done by including a validate method in Student class.

Modify the code written in Student Class to implement selection statements using if-else and logical operators

Modified Student class diagram:



Step 1: Modify the **Student** class created (Student.java) in Assignment No. 3 of Day 2 to include the method whose prototype is given above

Step 2: The implementation of the method is as follows:

a. ***validateExamMarks()***
   i. If the ***qualifyingExamMarks*** is greater than or equal to 65 and less than or equal to 100, return ***true***
   ii. If not, return ***false***

Step 3: Add the following statements to the existing main method written in Demo class (Demo.java)

a. Invoke the ***validateExamMarks()*** method
b. If the ***qualifyingExamMarks*** are valid,
   a. Display "Valid Marks"
   b. If not valid, display "Invalid marks, the range of the qualifying exam marks is between 65 and 100"

Step 4: Compile the program, fix the errors if any

Step 5: Execute the program and verify the output

Estimated time: 15 minutes

Summary of this assignment:
In this assignment, you have learnt

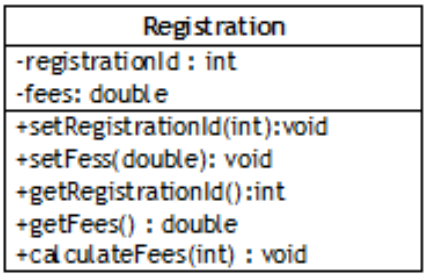
- Control Statements, operators and Boolean data type

Assignment 5: Control Structures

Objective: Creation of a Registration Class and instantiation of objects

Problem Description: Every student has to register and pay appropriate fees at the start of every academic year of engineering. The fees to be paid is calculated based on branch selected and discount based on qualification marks.

Implement the following class diagram using Java.



Step 1: Create a file called **Registration.java** using TextPad

Step 2: Define the class ***Registration*** as per the class diagram specified

Step 3: Define all the getter and setter methods of Registration Class

Step 4: The implementation of ***calculateFees()*** method is done based on the marks which is passed as an argument to the method. The details are given in Table 1:

Table 1:

Range of marks	Discount in %
85-100	12
75-84	7
65-74	0

i. Declare a local variable with the name **discount** of type integer to store the discount identified as per the **Table 1**
ii. Calculate the ***fees(instance variable of Registration class)*** after giving the discount as follows:

fees = ***fees*** - (***fees*** * (***discount*/100**))

Step 4: Write a class called ***DemoReg*** in a file **DemoReg.java** with the main method

Step 5: Create a reference variable of ***Registration*** class with the name ***reg*** and instantiate the same

a. Invoke the corresponding setter method to set the value for the instance variable as follows:

RegistrationId	2001

b. The fees is based on the Branch Id. Create a local variable called **branchId** in the main method and initialize to 1002. The logic for calculation of fees as per Branch Id is given in **Table 2**

Table 2

branchId	Fees
1001	25575.0
1002	15500.0
1003	33750.0
1004	8350.0
1005	20500.0

Write the logic for identifying the fees based on Branch Id using a switch case. (In the assignments of Day 3 we will accept the branch Id as an argument).Using the appropriate setter method, set the fees.

c. Invoke the calculateFees() with 79 as an argument for marks
d. Using the corresponding getter methods, and display the details as follows:

Registration Id :_____
Fees :_____

Step 5: Compile the program and execute the program

Step 6: Notice that the value of the discount calculated is not proper. Why?

Estimated time: 40 minutes

CONFIDENTIAL

Summary of this assignment: In this assignment, you have learnt

- To use operators and control structures to perform business logic

Assignment 6:Type Casting

Objective: Understand type casting

Problem Description: In Assignment 5 of Day 2, the discount was not getting calculated.

Step 1: Modify the formula written inside the ***calculateFees()*** as follows to get the right output:

fees = ***fees*** - (***fees*** * (***(double)discount***/**100**))

Compile and execute the program and you will notice that the discount calculation is taking place correctly.

Note:

- The discount calculation did not happen in Assignment 5 because the calculation discount/100 resulted in an integer value (there is loss of the precision part) and hence did not reflect in the output. In Assignment 6 the discount is type casted to double explicitly and hence the desired output is obtained

Step 2: Try out the same program with the below mentioned changes to the formula in ***calculateFees() :***

fees = ***fees*** - (***fees*** * (***discount***/**100.0**))

Does it work? If so,why?

Estimated time: 15 minutes

Summary of this assignment:

In this assignment, you have learnt

- Type Casting and Conversions

Assignment 7: Debugging - Type Casting

Objective: Understand type casting

CONFIDENTIAL

Problem Description: Debug the program created in Assignment 4 of Day 2. Perform the following steps:

Step 1: Modify the invocation of the setter method setQualifyingMarks() to pass 68.0 as the argument to the method.

Step 2: Compile the program, do you notice a compilation error , Why? What is the difference between the earlier invocation and the modified invocation?

Step 3: Rectify the error by using explicit type casting with argument, compile and execute the program

Note:

- Widening conversions are allowed whereas narrowing conversions result in compilation error
- It is not possible to assign a double to a float value

Estimated time: 10 minutes

Summary of this assignment:

In this assignment, you have learnt

- Type Casting and Conversions

Assignment 8: Loops

Objective: Understand loops

Problem Description: The code given below is written to display all the even numbers between 50 and 80 (both inclusive). Debug the program to get the correct output.

Step 1: Type the below program in a textpad, save the file as ForLoop.java, compile and execute.

```
public class ForLoop{
      public static void main(String args[]){
            for(int i=50;i<80;i++){
                  if(i%2==0){
                        System.out.println(i);
                  }
                  else{
```


CONFIDENTIAL

```
                        break;
                    }
                }
        }
}
```

Step 2: Correct the logical error in the code, save, compile and execute the code

Step 3: Implement the same logic using **while** loop

Estimated time: 20 minutes

Summary of this assignment:
In this assignment, you have learnt

- Usage of For loop, while loop and break statements

Assignment 9: Debugging - this reference

Objective: Understand usage of this reference to resolve instance variable hiding

Problem Description: Debug the below given code to get the output shown below after the code.

```
class Registration{
      private int registrationId;

      public void setRegistrationId(int registrationId){
            registrationId=registrationId;
      }

      public int getRegistrationId(){
            return registrationId;
      }
}

class Demo{
      public static void main(String args[]){
            Registration reg = new Registration();
            reg.setRegistrationId(1001);
            System.out.println("Registration Id:"+
```

```
                                                    reg.getRegistrationId());
      }
}
```

Step 2: Correct the logical error in the code, save, compile and execute the code to get the below output

Output Expected:

```
Registration Id: 1001
```

Estimated time: 10 minutes

Summary of this assignment:

In this assignment, you have learnt

- Usage of this reference for resolving the ambiguity between instance variables and local variables (instance variable hiding)

Self Review

Predict the output of the following programs and analyze your answer by executing the code. If there is an error, find the reason for the error and debug the same. **Also mention the inference drawn from each snippet provided in a notepad.**

1.

```
class TypeCasting{
     public static void main(String args[]){
        int intVal=10;
        double doubleVal= 20.0;
        intVal = doubleVal;
        System.out.println(intVal);
     }
}
```

2.

```
class Demo{
   public static void main(String args[]){
       if(1){
         System.out.println("True Block");
```

```
        }
        else{
            System.out.println("False Block");
       }
   }
}
```

3.

```
class Example{
      public static void main(String[] args){
            int intValOne = 0;
            int intValTwo = 0;
            for(short index=0; index < 5; index++){
                  if((++intValOne > 2) || (++intValTwo > 2)){
                        intValOne++;
                  }
            }
            System.out.println(intValOne + " " + intValTwo);
      }
}
```

4.

```
class Example{
      public static void main(String[] args){
            int intValOne = 5,intValTwo = 7;
            System.out.println(((  intValTwo * 2) % intValOne));
            System.out.println(intValTwo % intValOne);
      }
}
```

5.

```
class Example{
      public static void main(String[] args){
           int val1 = 9;
           int val2 = 6;
            for (int val3 = 0; val3 < 6; val3++,val2--){
            if(val1 > 2){
                        val1--;
            }
            if(val1 > 5){
                        System.out.print(val1 + " ");
```

```
                    --val1;
                    continue;
          }
          val1--;
        }
      }
}
```

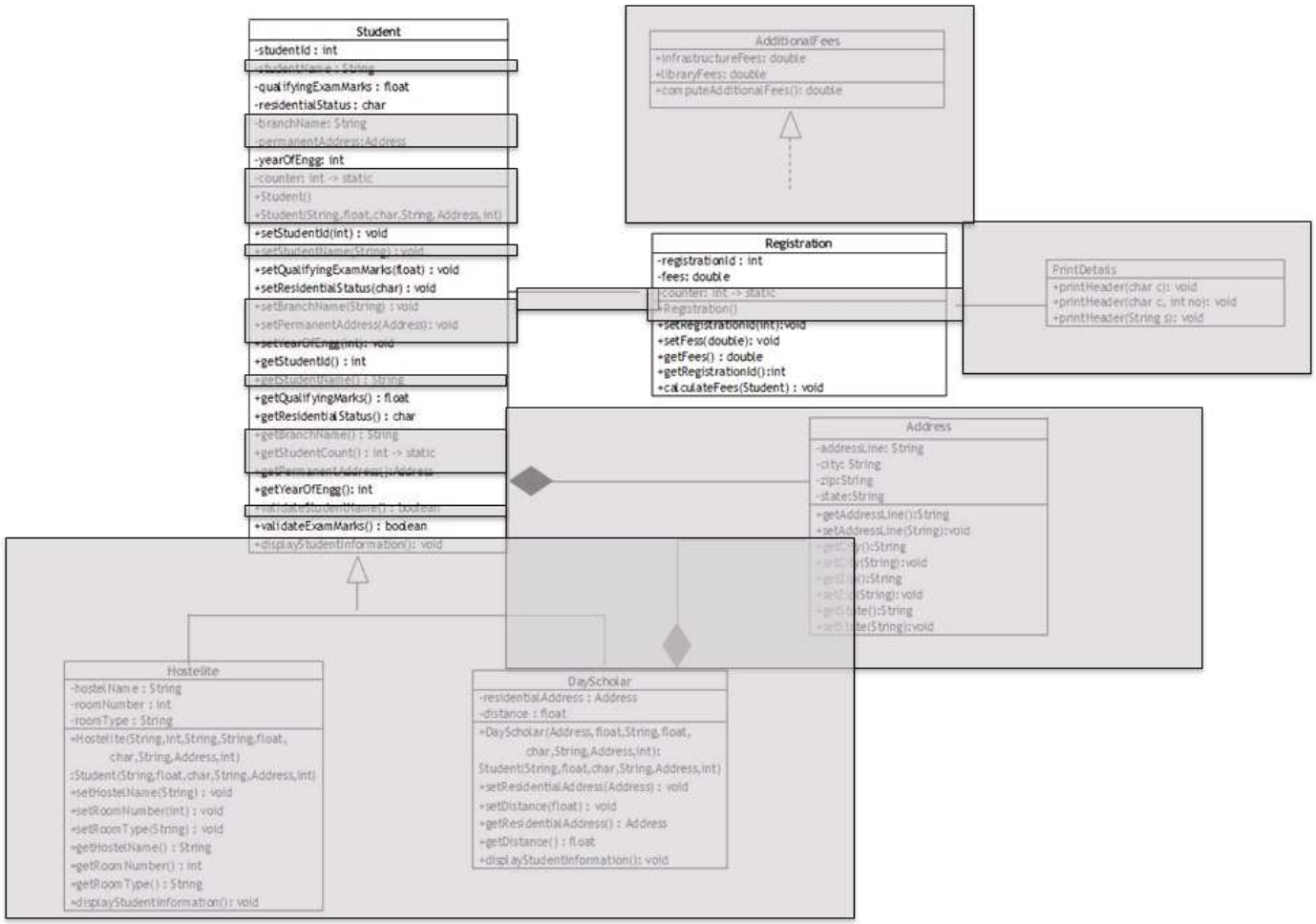
6.

```
      public class SwitchCase {
            public static void main(String args[]){
               int val = (int)(4 * 3);
               switch (val) {
                     case 0:  System.out.println("Case 0");
                              break;
                     case 1:  System.out.println("Case 1");
                              break;
                     case 2:  System.out.println("Case 2");
                              break;
                     default: System.out.println("Something is wrong ");
                  }
            }
```

CONFIDENTIAL

At the end of Day 2…..

The class diagram implemented so far….



At the end of Day 2 you have the following files:

Folder name	File name(s)
Assignment 1	Demo.java
Assignment 2	Demo.java
Assignment 3	Demo.java, Student.java
Assignment 4	Demo.java, Student.java
Assignment 5	DemoReg.java, Registration.java
Assignment 6	DemoReg.java, Registration.java
Assignment 7	Demo.java, Student.java
Assignment 8	ForLoop.java, WhileLoop.java

Context for Day 3

The following are the topics and concepts covered as part of the Day 3 assignments

- Arrays
- MultiDimensional and Jagged arrays
- Strings and methods inside the class
- Command Line Arguments
- Using a default constructor

Day 3 Assignments

Instructions:
Create a folder for Day 3 assignments in your home folder
Create a folder for each assignment and make the respective files related to that assignment in that folder.

Assignment 1: Arrays

Objective: Understand the concept of Arrays

Problem Description: In Day 2, we had several Branch Id's related to the various branches of engineering in the college. The fees to be paid is different for different branches. The business logic needs to be implemented to calculate the fees based on branch by using a data structure.

Create an array for the branch Id's and initialize them as follows:

Step 1: Open the file **DemoReg.java** with the main method written in Assignment 5 of Day 2. In the main method, do the following:

a. Declare an integer array ***branchList*** and initialize it with the following values:

branchList	1001	1002	1003	1004	1005

b. Declare a double array ***fees*** and initialize it with the following values:

fees	25575	15500	33750	8350	20500

Note: The ***branchList*** array and ***fees*** array are related as follows: the fees for branchId in ***branchList[0]*** is ***fees[0]***, the fees for branchId in ***branchList[1]*** is ***fees[1]*** and so on

CONFIDENTIAL

c. Check whether the value of the local variable, **branchId** is present in the ***branchList*** array
[Hint: make use of length property of the array]
   i. If present, identify the fees for the specified **branchId** from the ***fees*** array. Using the appropriate setter method, set the fees
      a. Invoke the calculateFees() with 79 as an argument for marks
      b. Using the corresponding getter methods, and display the details as follows:

      Registration Id :_____
      Fees :_____

   ii. If not, display a message as "Invalid branch Id"

Step 2: Compile the programs – **Registration.java, DemoReg.java**, fix the errors if any. Did you notice any difference in the way the array is declared and initialized compared to C language?

Step 3: Execute the program and verify the output

Estimated time: 20 minutes

Summary of this assignment:
In this assignment, you have learnt
- Declaration of an integer array, its initialization and accessing the array elements

Assignment 2: Multi dimensional Arrays

Objective: Understand 2D array and its working

Problem Description: Create a class ***DemoArray*** to add 2D arrays. Save it in DemoArray.java file. Write the business logic to display the sum of the matrices. Below mentioned are a few guidelines that can be followed:

Initialize the 2D arrays as follows:
```
int firstArrayMatrix[][]={{1,1,1},{1,1,1}};
int secondArrayMatrix[][]={{2,2,2},{2,2,2}};
```

Display the two matrices on the console as follows:
```
Matrix 1
```


CONFIDENTIAL

```
 1 1 1
 1 1 1

Matrix 2
 2 2 2
 2 2 2
```

Display the sum of the matrices:

```
Sum of the Matrices
 3 3 3
 3 3 3
```

Estimated time: 20 minutes

Summary of this assignment:
In this assignment, you have learnt

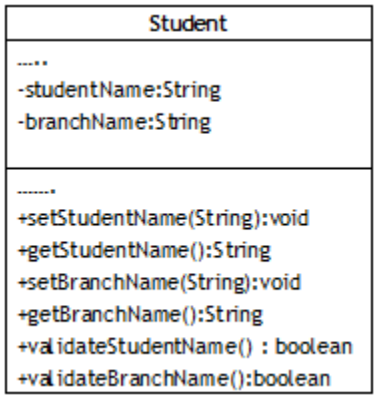
- Implementation of 2D arrays

Assignment 3: Strings

Objective: Write programs to understand the concept of Strings

Problem Description: The student name and the branch name details of a particular student has to be captured and validated.

Add the following mentioned in the class diagram of the existing Student class:



Step 1: Modify the ***Student*** class created (Student.java) in Assignment No. 4 of Day 2 to include the instance variables and the methods whose prototype is given above:

Step 2: The logic for the ***validateStudentName()***is as follows:

a. Check if the length of the ***studentName*** is between 5 and 25(both inclusive). If not, display a message "The length of the name should be minimum 5 characters and a maximum of 25 characters" and return false. Otherwise, return true.

Step 3: The logic for the ***validateBranchName()***is as follows:

a. Use the following table check whether the ***branchName*** is valid. Do a case insensitive comparison of Strings.

Table 3

branchName
CSE
ECE
EEE
MECH
Bio-Tech

If it is valid, return true, else set the value of ***branchName*** to CSE and display a message "Invalid Branch Name, set to CSE" and return false.

Step 4: Add the following statements to the existing main method written in ***Demo*** class (Demo.java in Assignment Number 4 of Day 2)

a. Create a reference variable of Student class(or use the existing object created earlier) with the name **studentOne** and instantiate the same

b. Invoke the corresponding setter methods to set the values for the instance variables as follows:

Table 4

studentId	1001
studentName	Jackson
qualifyingExamMarks	95.0f
residentialStatus	D
yearOfEngg	2
branchName	CSE

c. Invoke the ***validateStudentName()*** and ***validateBranchName()*** methods. If the return types of both the methods are true,

i. Invoke the ***validateExamMarks()*** method

ii. If the ***qualifyingExamMarks*** is invalid, display "Invalid marks, the range of marks is between 65 and 100"

iii. If the ***qualifyingExamMarks*** is valid,

CONFIDENTIAL

Using the corresponding getter method, display the details as follows:

Student Id :_____
Student Name :_____
Qualifying Marks :_____
Residential Status :_____
Current Year of Engineering :_____
Branch Name :_____

Step 5: Compile the program, fix the errors if any

Step 6: Execute the program and verify the output

Step 7: Create a reference variable of Student class with the name **studentTwo** and instantiate the same

a. Invoke the corresponding setter methods to set the values for the instance variables as follows:

Table 5

studentId	1002
studentName	Jen
qualifyingExamMarks	68.0f
residentialStatus	H
yearOfEngg	3
branchName	ABC

Repeat **Step 3.c** for **studentTwo** created in **Step 7**, compile and execute the program. Do you find a difference in the output? Why?

Estimated time: 30 minutes

Summary of this assignment:
In this assignment, you have learnt

- Using String data type

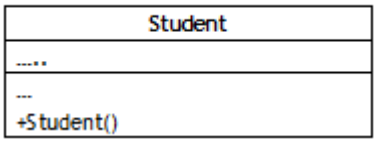
Assignment 4a: Default Constructor

Objective: Understand default constructor

CONFIDENTIAL

Problem Description: Constructors can be redefined by the programmer to perform initialization of instance variables and static variables of a class. Implement the default constructor for the Student class.

Modify the ***Student*** class created (Student.java) in Assignment No. 3 of Day 3 to add the following mentioned in the class diagram*:*



Step 1: Modify the ***Student*** class created (Student.java) of Assignment No. 3 of Day 3 to include the default constructor

Step 2: Add the following statements to the existing main method written in ***Demo*** class (Demo.java)

a. Create a reference variables of Student class with the name ***studentTemp*** and instantiate the same.

b. Using the corresponding getter method in the class to display the details as follows for the object:

Student Id :_____
Student Name :_____
Qualifying Marks :_____
Residential Status :_____
Current Year of Engineering :_____
Branch Name :_____

Step 3: Compile the program, fix the errors if any

Estimated time: 10 minutes

Summary of this assignment:
In this assignment, you have learnt

- Default constructor

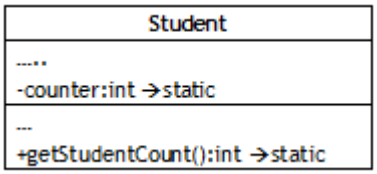
Assignment 4b: Default Constructor and static

CONFIDENTIAL

Objective: Understand default constructor

Problem Description: The student id is to be auto-generated by the system whenever a student joins the college. The Admin needs to keep track of the total number of students who have joined the college. Implement the auto-generation logic.

Modify the ***Student*** class created (Student.java) in Assignment No. 3 of Day 3 to add the following mentioned in the class diagram:



Step 1: Modify the ***Student*** class created (Student.java) of Assignment No. 3 of Day 3 to include the static variable

Step 2: Initialize the counter to 1000 **(Hint: you may use a static block for this)**

Step 3: Increment the value for counter by one and initialize the value of Student Id .

Step 4: Define a static method ***getStudentCount()*** which returns the number of students enrolled.(Hint: Use the static variable)

Step 5: Add the following statements to the existing main method written in ***Demo*** class (Demo.java)

a. Create two reference variables of Student class with the name ***studentThree*** and ***studentFour*** instantiate the same.

b. Invoke the corresponding setter methods to set the values for the instance variables as follows(***studentThree)***:

Table 6

studentName	Johan
qualifyingExamMarks	95.0f
residentialStatus	D
yearOfEngg	2

CONFIDENTIAL

branchName	CSE

c. Invoke the ***validateStudentName()*** and ***validateBranchName()*** for ***studentThree*** object created. If the return types of both the methods are true,

   i. Invoke the ***validateExamMarks()*** method for the ***studentThree*** object.
   ii. If the ***qualifyingExamMarks*** are invalid, display "Invalid marks, the range of marks is between 65 and 100"
   iii. If the ***qualifyingExamMarks*** are valid, using the corresponding getter method in the class to display the details as follows for both the objects:

Student Id :_____
Student Name :_____
Qualifying Marks :_____
Residential Status :_____
Current Year of Engineering :_____
Branch Name :_____

d. Invoke the corresponding setter methods to set the values for the instance variables as follows(***studentFour)***:

studentName	James
qualifyingExamMarks	73.0f
residentialStatus	D
yearOfEngg	1
branchName	MECH

e. Repeat **Step 5.c** for ***studentFour***

Step 6: Display the total number of students enrolled
Total number of students :_____

Step 7: Compile the program, fix the errors if any

Step 8: Execute the program and verify the output. Did you notice that the ***studentId*** is incremented by one for each of the objects created? Can the ***studentId*** be made static? If yes, Why? If not, Why?

CONFIDENTIAL

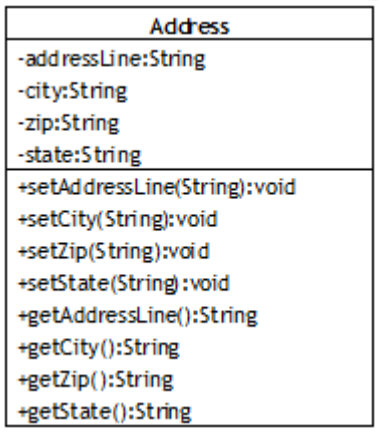
Estimated time: 25 minutes

Summary of this assignment:
In this assignment, you have learnt
- Default constructor
- Static block , static keyword and auto-generation

Assignment 5: Strings

Objective: Understand String data type

Problem Description: The permanent address and the residential address of the students need to be captured. For this purpose, an Address class was identified on Day 1. Implement the Address class as per the class diagram given below:



Step 1: Create a file Address.java and create the ***Address*** class as shown in the class diagram

Step 2: Implement the getter and setter methods

Step 3: Create a file called DemoAddress.java and create a class called ***DemoAddress*** containing the main method

Step 4: Create a reference variable of type ***Address*** and initialize the values of the instance variables as given in table below using the appropriate setter methods:

Table 6

addressLine	No.3,KT street
city	Mysore
zip	570001
state	Karnataka

Step 5: Use the corresponding getter methods display the details in the following format:

Address Line : _____
City : _____
Zip code : _____
State : _____

Step 6: Compile the program, fix the errors if any

Estimated time: 20 minutes

Summary of this assignment:
In this assignment, you have learnt

- Using String data type

Assignment 6: Command Line Arguments

Objective: Usage of command line arguments and integer conversion

Problem Description: The branch chosen by the student can be accepted by the program as an argument. Implement the same using command line argument.

Use the **DemoReg.java** used in Assignment 1 of Day 3 to include command line arguments.

Step 1: In the ***DemoReg*** class remove the initialization for the branchId local variable

Step 2: Accept the branch Id as a command line argument. Initialize the branch Id to the value of the command line argument after appropriate conversion (Hint: Use Integer.parseInt()).

Step 3: The rest of the program remains the same. Compile the programs – **Registration.java and DemoReg.java** and execute the code. Include the command line arguments. For details of including the command line arguments with Text Pad refer the **Passing Command line arguments using TextPad.pdf** supplied with the lab guide.

Estimated time: 20 minutes

Summary of this assignment: In this assignment, you have learnt

- Command line arguments and their usage

CONFIDENTIAL

Assignment 7: Multidimensional Array(Jagged Array)

Objective: Understand the concept of Jagged Array

Problem Description: Write a program to implement an asymmetric two dimensional array of Strings

Consider that the students joined a tutorial for various subjects as given below:

Table 7

Student Name	Tutorial1	Tutorial 2	Tutorial 3
Tony	JAVA	C	C++
Thomas	JAVA	UNIX	
Dinil	Linux	Oracle	
Delvin	RDBMS	C#	ORACLE

Step 1: Create a file JaggedArray.java with a class ***JaggedArray*** containing the main method

Step 2: Initialize the array based on the above table and display the tutorials attended by Delvin.

Estimated time: 20
Summary of this assignment:

In this assignment, you have learnt
- Multidimensional and asymmetric two dimensional Array

Note:
- In C language, an array of strings is represented by a two-dimensional array. In Java, array of strings is a single dimensional array consisting of instances of String class

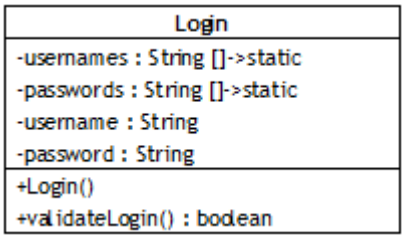
Assignment 8: Array of Strings

Objective: Understand an array of Strings

Problem Description: The Admin who is using the system has to be validated using a username and password at the time of logging into the system. Create a login class for the same.

CONFIDENTIAL

Step 1: Create a class called ***Login*** in Login.java file as per the below mentioned class diagram



Step 2: Initialize the ***usernames*** and ***passwords*** arrays with the values given below in a static block:

Table 8

usernames	admin	finance	student
passwords	Admin	Finance	Student

Step 3: In the default constructor, initialize the instance variables as given:

Table 9

Instance Variable	Value to be initialized
username	student
password	Student

Step 4: Implement the ***validateLogin***() method as below:

a. A case insensitive comparison should be made for the instance variable ***username*** against the usernames present in the ***usernames*** array
b. A case sensitive comparison should be made for the instance variable ***password*** against the passwords present in the ***passwords*** array and it should be the password corresponding to the username
c. If the username and password are valid, return ***true***
d. If not, return ***false***

Step 6: Create a file DemoLogin.java containing a class DemoLogin which in turn has the main method. In the main method

a. Create a reference variable of ***Login*** class with the name **login**
b. Invoke the ***validateLogin()*** method
   i. If the return value is ***true***, display a message "Login Successful"
   ii. If not, display an error message "Login Denied"

Step 8: Compile and execute the code(compile the classes individually) .

Estimated time: 30 minutes

CONFIDENTIAL

Summary of this assignment:

In this assignment, you have learnt

- How to declare, initialize and use the Array of Strings
- equals() and equalsIgnoreCase() methods of String class

Self Review

Predict the output of the following programs and analyze your answer by executing the code. If there is an error, find the reason for the error and debug the same. **Also mention the inference drawn from each snippet provided in a notepad.**

1.

```
class StringEqual{
      public static void main(String arg[]){

        String name1,name2;
        name1 = "God";
        name2= name1;
        if(name1 == name2){
            System.out.println("Equal");
        }
        else{
            System.out.println("Not Equal");
        }
    }
}
```

2.

```
class StringEqual{
      public static void main(String arg[]){

        String name1,name2;
        name1 = new String("God");
        name2= new String("God");
        if(name1 == name2){
            System.out.println("Equal");
        }
        else{
            System.out.println("Not Equal");
        }
    }
```

```
}
```

3.

```
class StringEqual{
      public static void main(String arg[]){

        String name1,name2;
        name1 = new String("God");
        name2= new String("God");
        if(name1.equals(name2)){
            System.out.println("Equal");
        }
        else{
            System.out.println("Not Equal");
        }
    }
}
```

4.

```
class StaticDemo
{
  static int count=10;

      StaticDemo(){
        count++;
      }
      static void display(){
        System.out.println(count);
      }

      static{
       System.out.println("Static Block written Before Main");
      }

     public static void main(String args[]){
         StaticDemo t1=new StaticDemo();
         StaticDemo t2=new StaticDemo();

         StaticDemo.display();
         t2.display();

       }
       static{
             System.out.println("Static Block written After Main");
           }
```

```
}
```

5.

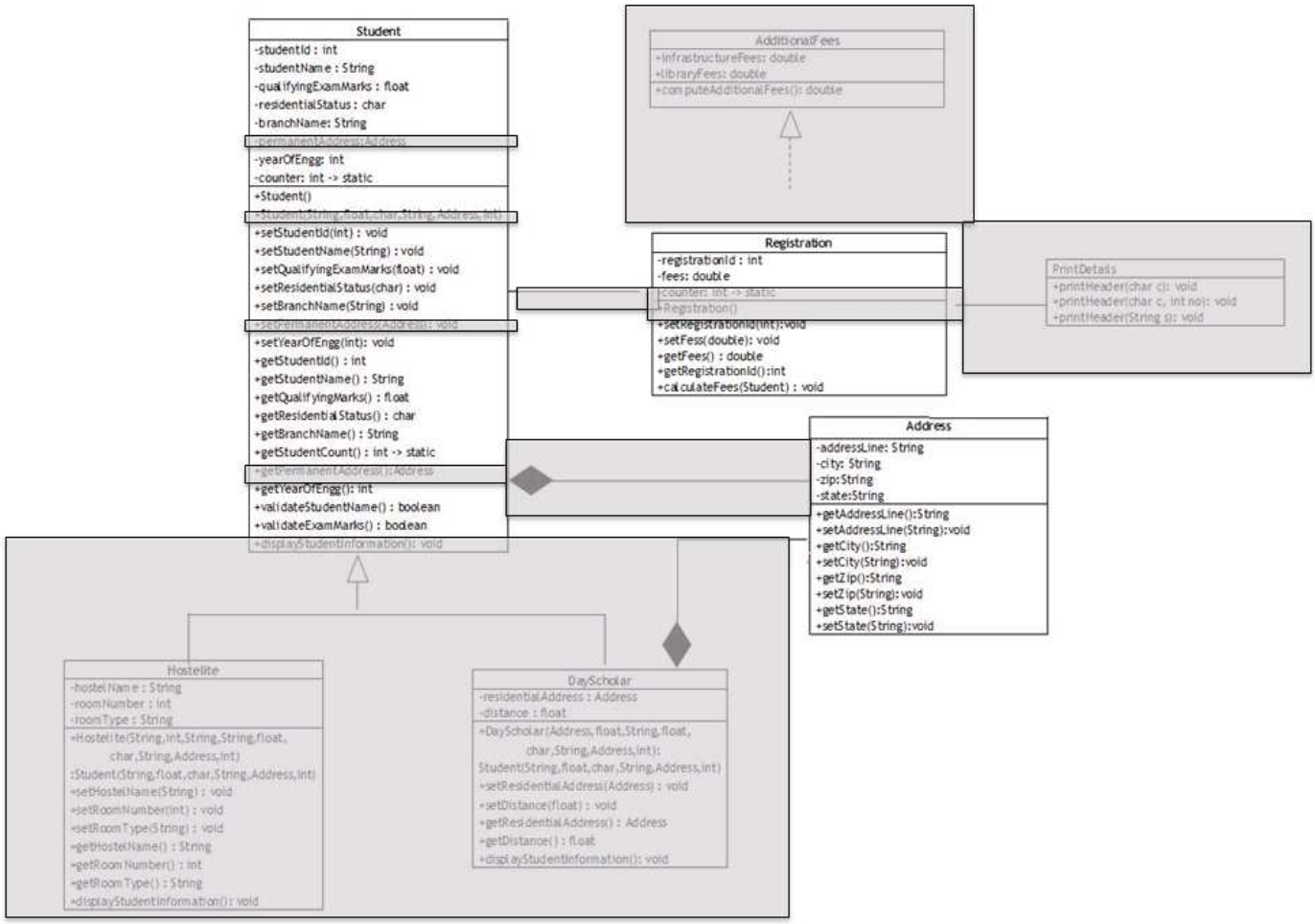
```
class MultiDimArray {
    public static void main(String[] args) {
        String[][] names = {{"Mrs.","Ms." , "Mr."},
                            {"Leo","James","John","Lena"}};
        System.out.println(names[0][0] + names[1][0]);
        System.out.println(names[0][2] + names[1][2]);
    }
}
```

6.

```
class StaticDemo1{
      private int num;
      static void display(){
            System.out.println(num);
      }


      public static void main(String args[]){
            StaticDemo1.display();
      }
}
```

At the end of Day 3…..

The class diagram implemented so far….



At the end of Day 3 you have the following files

Folder name	File name(s)
Assignment 1	DemoReg.java, Registration.java
Assignment 2	DemoArray.java
Assignment 3	Demo.java, Student.java
Assignment 4a	Demo.java, Student.java
Assignment 4b	Demo.java, Student.java
Assignment 5	DemoAddress.java, Address.java
Assignment 6	DemoReg.java, Registration.java
Assignment 7	JaggedArray.java
Assignment 8	Login.java, DemoLogin.java

Context for Day 4

The following are the topics and concepts covered as part of the Day 4 assignments

- Method Overloading
- Constructors - Parameterized Constructors
- Inheritance
- Aggregation
- Association
- Dynamic Polymorphism

Day 4 Assignments

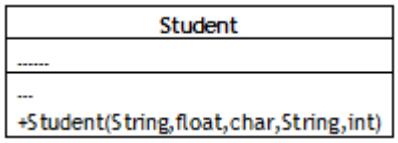
Instructions:
Create a folder for Day 4 assignments in your home folder
Create a folder for each assignment and make the respective files related to that assignment in that folder.

Assignment 1: Parameterized constructor and Constructor Overloading

Objective: Understand parameterized constructor

Problem Description: Parameterized constructors can be used to initialize the instance variables of the class. This can be done by passing arguments at the time of object creation.

Modify the ***Student*** class created (Student.java) on Day 3 to add the following mentioned in the class diagram:



Step 1: Modify the ***Student*** class created (Student.java) of Assignment No. 4 of Day 3 to include the definition for the parameterized constructor as shown in above class diagram.

Step 2: Initialize the values of the instance variables appropriately using the values passed to the parameterized constructor.
(Note: Ensure that the Student Id gets generated every time the default or the parameterized constructor is called)

CONFIDENTIAL

Step 3: Add the following statements to the existing main method written in ***Demo*** class (Demo.java)

a. Create a reference variable of Student class with the name ***studentFive*** by invoking the parameterized constructor of the student class by passing the following values:

Table 10

studentName	Jimmy
qualifyingExamMarks	688.0f
residentialStatus	H
yearOfEngg	2
branchName	ECE

b. Invoke the ***validateStudentName()*** and ***validateBranchName()*** for the object created. If the return types of both the methods are true,

i. Invoke the ***validateExamMarks()*** method using the object created.
ii. If the ***qualifyingExamMarks*** are invalid, display "Invalid marks, the range of marks is between 65 and 100"
iii. Else, using the corresponding getter methods in the class to display the details as follows for the object created:

Student Id :_____
Student Name :_____
Qualifying Marks :_____
Residential Status :_____
Current Year of Engineering :_____
Branch Name :_____

Step 6: Display the total number of students enrolled

Total number of students :_____

Step 7: Compile and Execute the program and verify the output. Did you notice that the ***studentId*** is incremented by one for each of the object created? Did you also notice that there are two versions of the constructor?

CONFIDENTIAL

Note:
- The setter methods may be used for modification of instance variable values
- The setter may also be used for setting the values of the instance variables in the absence of a parameterized constructor
- Constructors can be overloaded

Estimated time: 20 minutes

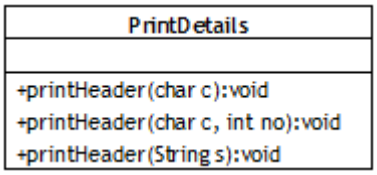
Summary of this assignment:
In this assignment, you have learnt
- Parameterized constructor

Assignment 2: Method Overloading

Objective: Understand method overloading

Problem Description: The Admin wants to print the header of the bill in multiple ways depending on the need. This can be printing a single character n, number of times or printing of String or printing a character specified number of times.

Create a class called ***PrintDetails*** with overloaded methods as shown in the class diagram*:*



Step 1: Create a file called **PrintDetails.java** which contains the **PrintDetails** class designed as per the class diagram given above.

Step 2: Implement the **printHeader()** methods as described below:

a. ***printHeader(char c)***
   - This method would print the character passed to it as an argument 80 times in the same line

[Hint: make use of System.out.print() method to retain the cursor on the same line]

b. ***printHeader(char c, int no)***

CONFIDENTIAL

- This method would print the character passed to it as an argument the specified number of times, which is passed as a second argument, in the same line

c. ***printHeader(String s)***

- This method would print the string passed to it as an argument once

Step 3: Create a file **DemoPrint.java** containing the ***DemoPrint*** class with the main method.

Step 4: Create an object of ***PrintDetails*** class and instantiate the same. Invoke the ***printHeader*** methods with the following values for the arguments

Table 11

Variables	values
c	*
no	20
s	Ricto Engineering College

Step 7: Compile , execute the program and verify the output

Note:

- Methods can be overloaded based on the number, sequence and type of arguments

Estimated time: 25 minutes

Summary of this assignment:

In this assignment, you have learnt

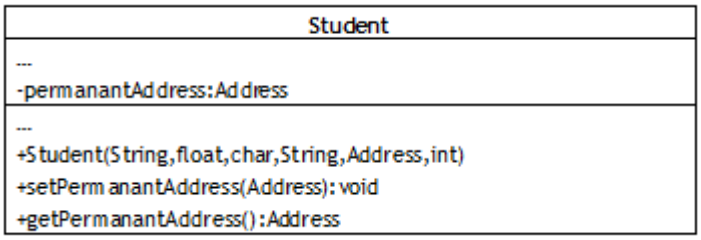
- Method Overloading

Assignment 3: Relationships – 'has-a' relationship

Objective: Understand has-a relationship

Problem Description: The relationship between the ***Student*** and the ***Address*** class is that of 'has-a' relationship. Using a java program, the relationship between the two classes can be established by holding a reference of the ***Address*** class inside ***Student*** class as every student has an address.

CONFIDENTIAL



Step 1: Use the Student.java file created in Assignment 1 of Day 4. Modify the class by adding an instance variable of type Address. (Use the Address class created in Assignment 5 of Day 3)

Step 2: Modify the parameterized constructor to add the ***Address*** reference as a parameter and initialize the ***permanantAddress*** with the reference passed

Step 3: The definition of setPermanantAddress() and getPermanantAddress() is as given below:

```
class Student{
      ........
      private Address permanantAddress;
      ..........
     public void setPermanantAddress(Address address){
            permanantAddress= address;
     }
     public Address getPermanantAddress(){
           return permanantAddress;
      }
      ........
}
```

Step 4: Create a file **DemoConstructor.java** file with the main method

```
class DemoConstructor{
 public static void main(String args[]){
      Address address = new Address();
      address.setAddressLine("No.333,ABC street");
      address.setCity("Mysore");
     address.setState("Karnataka");
     address.setZip("570001");
     Student studentSix = new Student();
     studentSix.setPermanantAddress (address);
     //Write the code for invocation of the setter methods, validate
```

```
      //methods and printing the other details in the Student class as
      //per previous the assignments done
      address =studentSix.getPermanantAddress();
      System.out.println("Address:");
      System.out.println(address.getAddressLine()+",");
      System.out.println(address.getCity()+",");
      System.out.println(address.getState()+",");
      System.out.println(address.getZip());
 }
}
```

Step 5: Copy the **Address.java** file into the present working folder, compile the code. Compile the **Student.java** and also compile the **DemoConstructor.java** program and fix errors if any

Step 6: Execute the program and verify the output

Note:

- The reference of the Address class is placed as a member of the Student class representing aggregation

Estimated time: 25 minutes

Summary of this assignment:

In this assignment, you have learnt

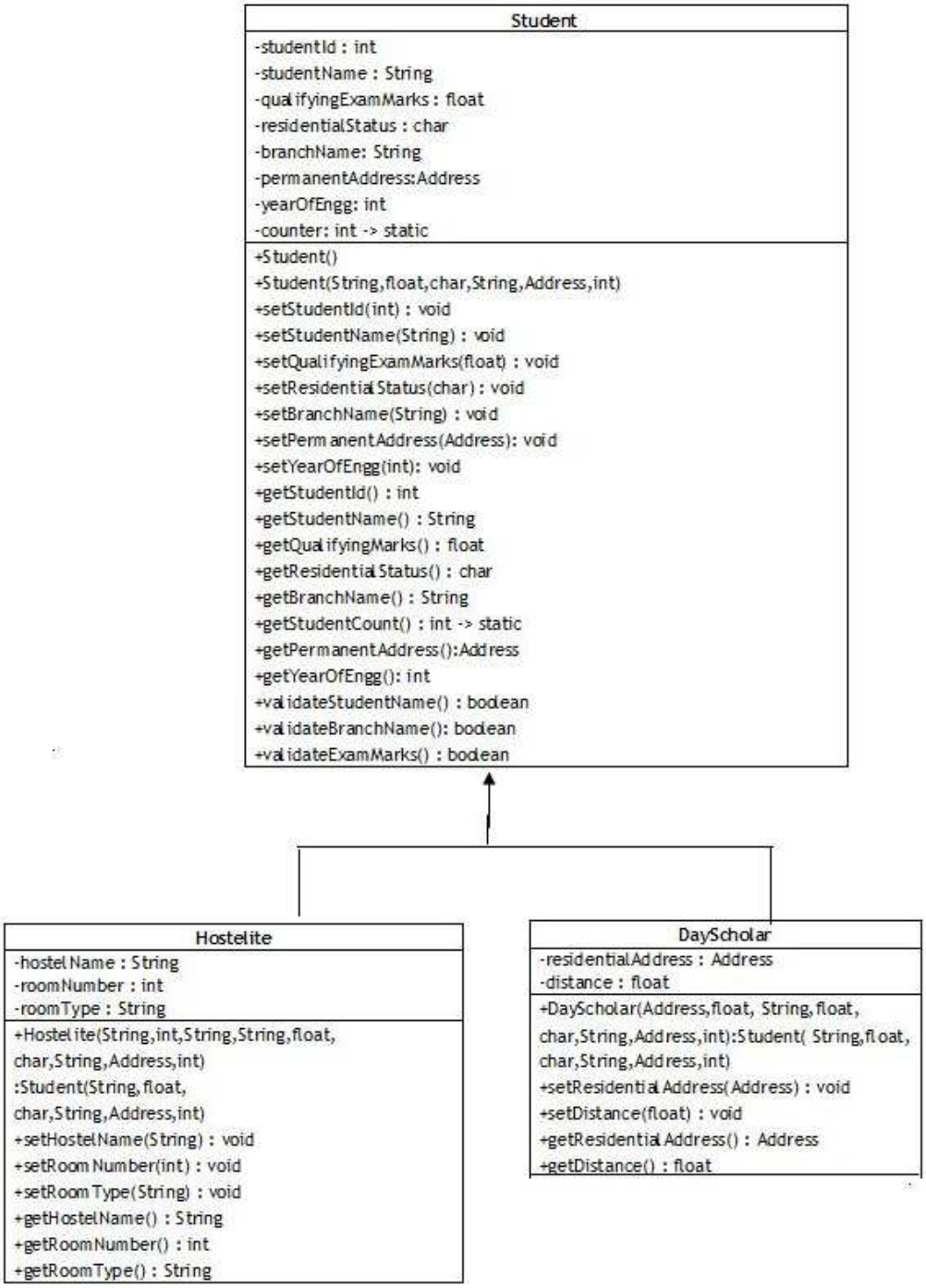
- Aggregation relationship

Assignment 4: Relationships - 'is a' relationship

Objective: Understand the concept of is-a relationship

Problem Description: ***Hostelite*** and ***DayScholar*** are two classes which inherit from the ***Student*** class. They represent the two different kinds of ***Student***. The class diagram is as given below:

CONFIDENTIAL


Student
-studentId : int
-studentName : String
-qualifyingExamMarks : float
-residentialStatus : char
-branchName: String
-permanentAddress:Address
-yearOfEngg: int
-counter: int -> static
+Student()
+Student(String,float,char,String,Address,int)
+setStudentId(int) : void
+setStudentName(String) : void
+setQualifyingExamMarks(float) : void
+setResidentialStatus(char) : void
+setBranchName(String) : void
+setPermanentAddress(Address): void
+setYearOfEngg(int): void
+getStudentId() : int
+getStudentName() : String
+getQualifyingMarks() : float
+getResidentialStatus() : char
+getBranchName() : String
+getStudentCount() : int -> static
+getPermanentAddress():Address
+getYearOfEngg(): int
+validateStudentName() : boolean
+validateBranchName(): boolean
+validateExamMarks() : boolean
Hostelite
-hostelName : String
-roomNumber : int
-roomType : String
+Hostelite(String,int,String,String,float,
char,String,Address,int)
:Student(String,float,
char,String,Address,int)
+setHostelName(String) : void
+setRoomNumber(int) : void
+setRoomType(String) : void
+getHostelName() : String
+getRoomNumber() : int
+getRoomType() : String
DayScholar
-residentialAddress : Address
-distance : float
+DayScholar(Address,float, String,float,
char,String,Address,int):Student(String,float,
char,String,Address,int)
+setResidentialAddress(Address) : void
+setDistance(float) : void
+getResidentialAddress() : Address
+getDistance() : float

Step 1: Create a new file **Hostelite.java** and define the ***Hostelite*** class which extends the **Student** class as per the above class diagram

Step 2: Implement the constructor of ***Hostelite*** class and invoke the constructor of ***Student*** class using the ***super*** keyword

Step 3: Compile the **Student.java** and **Hostelite.java** file and fix the errors if any

Step 4: Create a new file **DayScholar.java** and define the ***DayScholar*** class which extends the **Student** class as per the above class diagram

Step 5: Implement the constructor of ***DayScholar*** class and invoke the constructor of ***Student*** class using the ***super*** keyword

Step 6: Compile the **DayScholar.java** file and fix the errors if any

Step 7: Create a new file **DemoInherit.java** containing the ***DemoInherit*** class with the main method

Step 8: In the main method:

a. Create a reference variable of ***Hostelite*** class and instantiate the same with the following values using the parameterized constructor as given below

Table 12

Instance Variable	Value to be initialized
hostelName	Aaradhya
roomNumber	105
roomType	double
studentName	Tommy
qualifyingExamMarks	75.6f
residentialStatus	H
branchName	CSE
yearOfEngg	4
addressLine	L-51,SriLabdhi Colony
city	Chennai
state	TamilNadu
zip	600004

(Hint: Create reference variables of Address class whenever an address needs to be stored)

CONFIDENTIAL

b. Invoke the ***validateStudentName()*** method. If the name is valid, invoke the ***validateBranchName()*** and ***validateExamMarks()*** method. If both are valid, invoke the corresponding getter methods and display the following details :

Student Id :_____
Student Name :_____
Qualifying Marks :_____
Residential Status :_____
Current Year of Engineering :_____
Branch Name :_____
Permanent Address :_____
Hostel Name :_____
Room Number :_____
Room Type :_____

c. Create a reference variable of ***DayScholar*** class and instantiate the same with the following values:

Table 13

Instance Variable	Value to be initialized
For Residential Address	
addressLine	No.3, Choultry circle
City	Mysore
state	Karnataka
Zip	570017
distance	10
studentName	Timmy
qualifyingExamMarks	80.0f
residentialStatus	D
branchName	CSE
yearOfEngg	4
For Permanant Address	
addressLine	No.L-82, West Fort
City	Trivandrum
state	Kerala
Zip	560015

(Hint: Create reference variables of Address class whenever an address needs to be stored)

CONFIDENTIAL

a. Invoke the ***validateStudentName()*** method. If the name is valid, invoke the ***validateBranchName()*** and ***validateExamMarks()*** method. If both are valid, invoke the corresponding getter methods and display the following details :

Student Id :_____
Student Name :_____
Qualifying Marks :_____
Residential Status :_____
Current Year of Engineering :_____
Branch Name :_____
Residential Address :_____
Distance :_____
Permanent Address :_____

Step 11: Compile, execute the program and verify the output

Estimated time: 40 minutes

Summary of this assignment:
In this assignment, you have learnt

- Inheritance

Assignment 5: Relationships - Uses a relationship

Objective: To understand the concept of Uses-A relationship

Problem Description: The relationship between ***Registration*** class and the ***PrintDetails*** class is that of association. The ***calculateFees()*** method uses **PrintDetails** class for printing the header of the bill. Implement this relationship using a Java program.

Step 1: Use the file **Registration.java** created on Day 2, Assignment 6 and make the changes to the ***calculateFees()*** before the calculation of fees is done

a. Create a reference variable of ***PrintDetails*** class
b. Invoke the ***printHeader()*** method of the ***PrintDetails*** class by passing a character '*' to it
c. Invoke the ***printHeader()*** method of the ***PrintDetails*** class by passing a String " Ricto Engineering College" to it
d. Invoke the ***printHeader()*** method of the ***PrintDetails*** class by passing a character '*' to it
e. Display the ***registrationId*** as shown below:

CONFIDENTIAL

Registration Id: _____

Step 2: The rest of the implementation of the ***calculateFees()*** method remains the same .Compile the file **Registration.java**

Step 3: Use the class ***DemoReg*** created in the file **DemoReg.java** with the main method on Day 2. Remove the code written to display ***registrationId*** since it is now displayed in ***calculateFees()*** of **Registration** class.

Step 4: Compile **DemoReg.java** and execute the code. Do you observe the printing of the header? What is the relationship between the Registration and the PrintDetails class? How is it different from aggregation?

Estimated time: 15 minutes

Summary of this assignment:
In this assignment, you have learnt

- The concept of Uses – A relationship

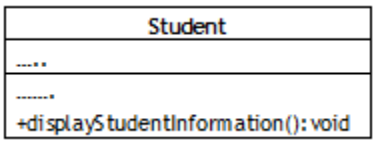
Assignment 6: Method Overriding

Objective: Understand method overriding

Problem Description: Method overriding happens when a method defined in the parent class is redefined in the derived classes. ***displayStudentInformation()*** has to be defined in the Student class to display student information and overridden in the derived classes (Hostelite and DayScholar) to display the respective class details.

Step 1: Use the file **Student.java, Address.java, Hostelite.java and DayScholar.java files** created earlier

Step 2: In the ***Student*** class, add a new method called ***displayStudentInformation()*** as shown in the class diagram below:



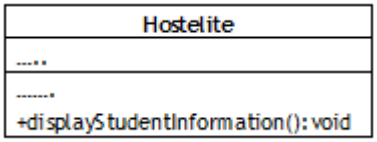
In the ***displayStudentInformation()***,write the logic using the getter methods of the class and display the details as follows:

Notice how within the method, there is no need for an object reference as the 'this' reference is implicitly available

```
public void displayStudentInformation(){
      System.out.println("Student Id          : "+getStudentId());
      System.out.println("StudentName         : "+getStudentName());
      System.out.println("Qualifying Marks    : "+getQualifyingMarks());
      System.out.println("Residential status:"+getResidentialStatus());
      System.out.println("Current Year Of Engg    : "+getYearOfEngg());
      System.out.println("Branch Name         : "+getBranchName());
      System.out.println("Address:"+getPermanantAddress().getAddressLin
e()+getPermanantAddress().getCity()+getPermanantAddress().getState()+g
etPermanantAddress().getZip());
      }
```

Notice how the getPermanantAddress() method returns a reference of Address class, which in turn is used to invoke a method of the Address class

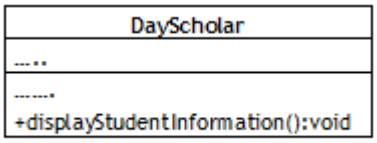
Step 2: Use the file **Hostelite.java** file created earlier, add a new method called ***displayStudentInformation()*** as shown in the class diagram below:



In the ***displayStudentInformation()***,invoke the ***displayStudentInformation()*** of the ***Student*** class .Write the logic using the getter methods of the **Hostelite** class and display the details as follows:

```
public void displayStudentInformation(){
            super.displayStudentInformation();
            System.out.println("Hostel Name         : "+getHostelName());
            System.out.println("Room Number         : "+getRoomNumber());
            System.out.println("Room Type     : "+getRoomType());
}
```

Step 3: Use the file **DayScholar.java** file created earlier, add a new method called ***displayStudentInformation()*** as shown in the class diagram below:



In the ***displayStudentInformation()***,invoke the ***displayStudentInformation()*** of the ***Student*** class .Write the logic using the getter methods of the **DayScholar** class and display the details as follows:

```
public void displayStudentInformation(){
            super.displayStudentInformation();
            System.out.println("Distance     : "+getDistance());
            System.out.println("Residential Address          :
"+getResidentialAddress().getAddressLine()+getResidentialAddress().get
City()+getResidentialAddress().getState()+getResidentialAddress().getZ
ip());
}
```

Step 4: Create a file **DemoOverriding.java** containing the ***DemoOverriding*** class with the main method.

Step 5: Create an object each of the ***Hostelite*** and ***DayScholar*** class and instantiate them. (Pass values for each of the instance variables through the parameterized constructor). Invoke the ***displayStudentInformation()*** using the objects individually.

Step 6: Compile the **Student.java, Hostelite.java, DayScholar.java** and **DemoOverriding.java** and execute the code. Verify the output.

Estimated time: 20 minutes

Summary of this assignment:
In this assignment, you have learnt
- Overriding of methods

Assignment 7: Dynamic Polymorphism

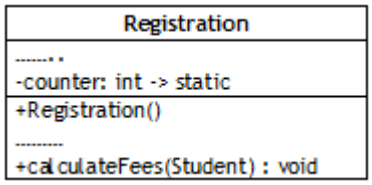
Objective: Understand dynamic polymorphism, implementation and applications

Problem Description: The calculation of the fees paid by a student is based on the following:
- Qualifying marks
- Hostelite or Day scholar

- Year of Engineering
- Additional fees to be paid

The ***calculateFees()*** of ***Registration*** class has to be modified based on whether the student is a hostelite or a day scholar.

Modify the Assignment 5 of Day 2 to implement the above.



Step 1: Use the Student.java, Hostelite.java, DayScholar.java and Address.java created earlier

Step 2: Use the **Registration.java** created earlier and add the following to the ***Registration*** class:

- A static counter variable
- Constructor of the ***Registration*** class
- Change the prototype of the **calculateFees()** to pass a **Student** reference to it

Step 2: Implementation of methods is given below:

a. In the constructor of ***Registration*** class, write the logic so that the registration id's are autogenerated from 5001 onwards.
b. In ***calculateFees()*** method, using the ***Student*** reference passed, do the following validations:
   i. Invoke the ***validateStudentName()*** method. If the name is valid, invoke the ***validateBranchName()*** and ***validateExamMarks()*** method. If both are valid, go to the next step (Step 2.c). Otherwise display appropriate error messages and return 0

A skeleton code is given for your reference:

```
public void calculateFees(Student stu){
            int discount=0;
            if(stu.validateStudentName() ==true){
                  if(stu.validateBranchName() ==true){
                        if(stu.validateExamMarks() ==true){
                              if(stu.getQualifyingMarks()>=85
&& stu.getQualifyingMarks()<=100){
```

```
                                        discount=12;
                                    }
……………………………….
}
```

c. Identify the fees based on the branch name of the student using the student reference as per the below table:

Table 14

branchName	branchFees
CSE	25575
ECE	15500
EEE	33750
MECH	8350
Bio-Tech	20500

d. Identify the qualifying marks of the student (using the student reference and getter method) and calculate the discount by declaring a local variable discount. (In Day 2, recall that the marks was passed as an argument to the ***calculateFees()***). The table for your reference is as follows:

Table 15

qualifyingMarks	Discount in %
85-100	12
75-84	7
65-74	0

e. Calculate the ***fees(instance variable of Registration class)*** after giving the discount as follows:

fees = ***fees*** - (***fees*** * (***discount***/**100**))

f. Invoke the ***displayStudentInformation()*** method using the reference of student.

Step 3: Create a new file **DemoDynamic.java** with the class ***DemoDynamic*** which contains the main method.

Step 4: In the main method:

a. Create a reference variable of ***Student*** class called ***studentDynamicOne*** and instantiate the same to an object of ***Hostelite*** class with the following values (Use the parameterized constructor of respective class):

Table 16

Instance Variable	Value to be initialized

CONFIDENTIAL

hostelName	Aaradhya
roomNumber	105
roomType	double
studentName	Tommy
qualifyingExamMarks	75.6f
residentialStatus	H
branchName	CSE
yearOfEngg	4
addressLine	L-51,SriLabdhi Colony
city	Chennai
state	TamilNadu
zip	600004

(Hint: Create reference variables of Address class whenever an address needs to be stored)

Do you notice here that a reference variable of a parent class is instantiated with an object of derived class. Thus a parent reference may point to an object of itself or its child classes which is one aspect of dynamic polymorphism

b. Create a reference variable of Registration class called ***registrationOne*** . Using this object invoke the ***calculateFees()*** by passing the ***studentDynamicOne*** as an argument.
c. Invoke the corresponding getter method to display the fees as shown below:

Fees to be paid by the Hostelite:_____

A sample skeleton of code is shown for your reference:

```
Student studentDynamicOne=new
     Hostelite("Aradhya",105,"double","Tommy",75.6f,'H',"CSE",add,4);
Registration registrationOne=new Registration();
registrationOne.calculateFees(studentDynamicOne);
System.out.println("Fees for Hostelite:"+registrationOne.getFees());
```

Notice parent reference pointing to child class

Notice passing of Student reference to calculateFees()

d. Create another reference variable of ***Student*** class called ***studentDynamicTwo*** and instantiate the same to an object of ***DayScholar*** class with the following values(Use the parameterized constructor of respective class) :

Table 17

Instance Variable	Value to be initialized
For Residential Address	
addressLine	No.3, Choultry circle
City	Mysore
state	Karnataka
Zip	570017
distance	10
studentName	Timmy
qualifyingExamMarks	80.0f
residentialStatus	D
branchName	MECH
yearOfEngg	4
For Permanant Address	
addressLine	No.L-82, West Fort
City	Trivandrum
state	Kerala
Zip	560015

(Hint: Create reference variables of Address class whenever an address needs to be stored)

e. Using this object invoke the ***calculateFees()*** by passing the ***studentDynamicTwo*** as an argument.
f. Invoke the corresponding getter method to display the fees as shown below:

Fees to be paid by the Day Scholar:_____

Step 5: Compile **Registration.java** and **DemoDynamic.java** files, execute the program and verify the output

Estimated time: 45 minutes

Summary of this assignment:
In this assignment, you have learnt

- Dynamic polymorphism

CONFIDENTIAL

Self Review

What would be output for the following programs? If there is an error in the program, analyze it, fix it and execute it. **Also mention the inference drawn from each snippet provided in a notepad.**

1.

```
class Parent{

      private int num;

      public  void setNum(int val){
            num = val;
      }

      public int getNum(){
            return num;
      }

      public void display(){
            System.out.println("Number :"+num);
      }

}
class Child extends Parent{

     private int val;
      public void setVal(int num){
            val = num;
      }

      public int getVal(){
            return val;
      }

      public void display(){
            System.out.println("Number :"+num);
            System.out.println("Value :"+val);
      }
}
class CourseMgmt{

      public static void main(String args[]){
       Child child;

       child = new Child();
```

```
        child.setNum(100);
        child.setVal(200);
        child.display();

      }
}
```

2.

```
class Parent{
      private int num;

      public  void setNum(int val){
            num = val;
      }

      public int getNum(){
            return num;
      }

      public void display(){
            System.out.println("Number :"+num);
      }

}

class Child extends Parent{

private int val;
      public void setVal(int num){
            val = num;
      }

      public int getVal(){
            return val;
      }

      public void display(){
            super.display();
            System.out.println("Value :"+val);
      }

}

class CourseMgmt{
```

```
      public static void main(String args[]){
       Child child;

       child = new Child();

       child.setNum(100);
       child.setVal(200);
       child.display();

      }
}
```

3.

```
class Animal {
    public static void testClassMethod() {
        System.out.println("The class method in Animal.");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Animal.");
    }
}

class Cat extends Animal {
    public static void testClassMethod() {
        System.out.println("The class method in Cat.");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Cat.");
    }

    public static void main(String[] args) {
        Cat myCat = new Cat();
        Animal myAnimal = myCat;
        Animal.testClassMethod();
        myAnimal.testInstanceMethod();
    }
}
```

4.

```
class Parent{

      private int num;

      Parent(int num){
```

```
            this.num = num;
      }

      public int getNum(){
            return num;
      }


}

class Child extends Parent{

private int val;

      Child(int num,int val){
            // To invoke super class constructor
            super(num);
            this.val = val;
      }
      public int getVal(){
            return val;
      }
}

class CourseMgmt{

      public static void main(String args[]){
       Child child;

       child = new Child(100,200);
       System.out.println("Parent: Num:" + child.getNum());
       System.out.println("Child: Val:" + child.getVal());


      }
}
```

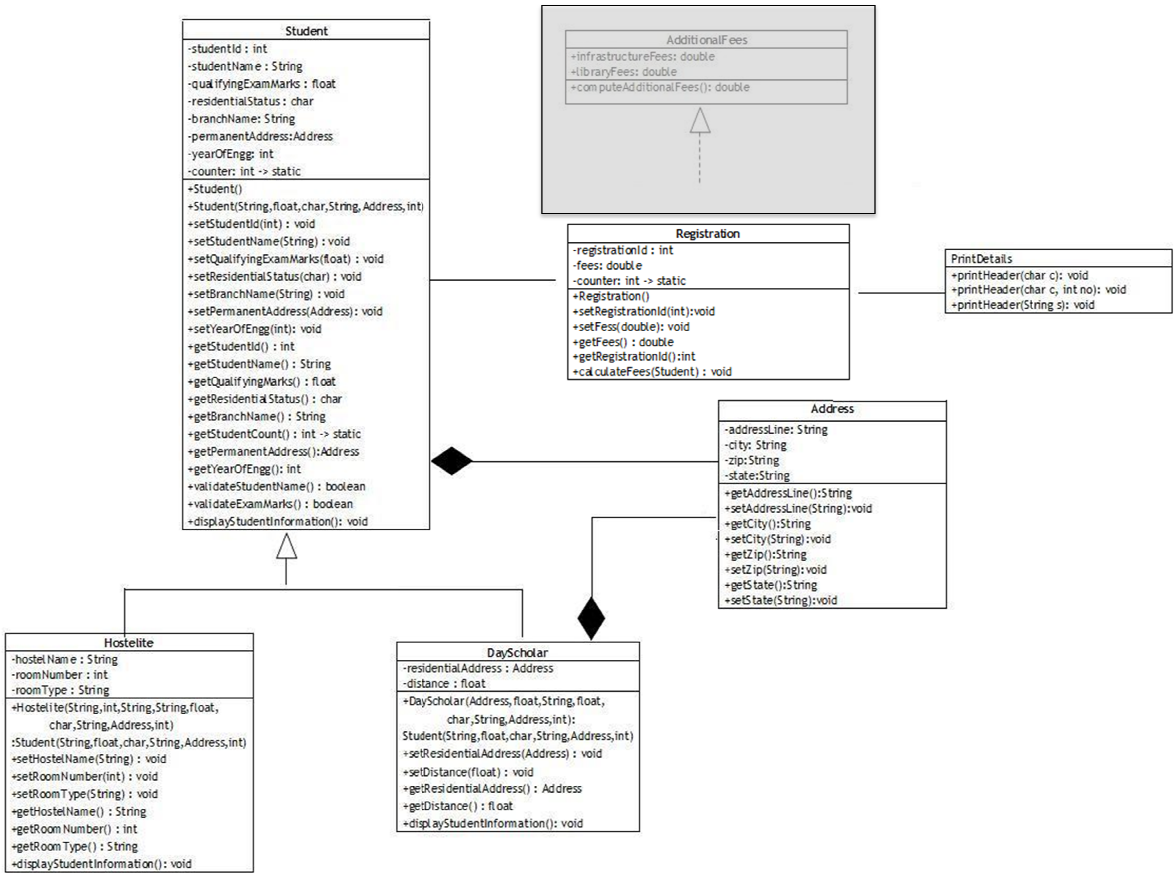
5.

```
public class ClassA {
    public void methodOne(int i) {
    }
    public void methodTwo(int i) {
    }
    public static void methodThree(int i) {
    }
    public static void methodFour(int i) {
```

```
    }
}

public class ClassB extends ClassA {
    public static void methodOne(int i) {
    }
    public void methodTwo(int i) {
    }
    public void methodThree(int i) {
    }
    public static void methodFour(int i) {
    }
}
```

At the end of Day 4…..

The class diagram implemented so far….



At the end of Day 4 you have the following files

Folder name	File name(s)
Assignment 1	Demo.java, Student.java
Assignment 2	DemoPrint.java, PrintDetails.java
Assignment 3	DemoConstructor.java, Student.java, Address.java
Assignment 4	DemoInherit.java, Student.java, Hostelite.java, DayScholar.java
Assignment 5	DemoReg.java, Registration.java, PrintDetails.java
Assignment 6	DemoOverriding.java, Student.java, Hostelite.java, DayScholar.java, Address.java
Assignment 7	DemoDynamic.java, Student.java, Hostelite.java, DayScholar.java, Address.java, Registration.java, PrintDetails.java

Context for Day 5

The following are the topics and concepts covered in the assignments as part of the Day 4 assignments

- ‘abstract‘ keyword
- ‘final’ keyword
- Interfaces
- Packages
- ‘abstract‘ keyword
- ‘final’ keyword

Day 5 Assignments

Instructions:
Create a folder for Day 5 assignments in your home folder
Create a folder for each assignment and make the respective files related to that assignment in that folder.
Also follow the instructions in the respective assignments while creating packages.

Assignment 1: abstract class and method

Objective: Understand abstract keyword for usage with class and method

CONFIDENTIAL

Problem Description: When at least one method in a class is not complete (ie. abstract) then, the class holding that method also needs to be declared abstract. The ***displayStudentInformation()*** is a good candidate for abstract method, as the same information can be accessed and displayed in the overridden method. The **Student** class, therefore has to be declared abstract.

Step 1: Use the file **Student.java, Address.java, Hostelite.java and DayScholar.java files** created in Assignment 6 of Day 4.

Step 2: Make the ***Student*** class and the ***displayStudentInformation()*** present in the class abstract. This is needed because the ***Hostelite*** and the ***DayScholar*** classes override this method. There is no definition to this method as shown below:

Notice the abstract method with no body

```
abstract class Student{
      ……..
      abstract public void displayStudentInformation();
}
```

Step 3: Use the file **Hostelite.java** file created earlier**.** In the ***displayStudentInformation()***, invoke the getter methods of the ***Student*** class explicitly (remove the invocation of the ***displayStudentInformation()*** using super) to display the values of the instance variables of the ***Student***. This is followed by the display of details of the ***Hostelite*** class members as shown below:

```
class Hostelite{
……….
public void displayStudentInformation(){
      System.out.println("Student Id        : "+getStudentId());
      System.out.println("StudentName       : "+getStudentName());
      System.out.println("Qualifying Marks  : "+getQualifyingMarks());
      System.out.println("Residential status:"+getResidentialStatus());
      System.out.println("Current Year Of Engg    : "+getYearOfEngg());
      System.out.println("Branch Name       : "+getBranchName());
      System.out.println("Address:"+getPermanantAddress().getAddressLine
      ()+getPermanantAddress().getCity()+getPermanantAddress().getState(
      )+getPermanantAddress().getZip());
      System.out.println("Hostel Name       : "+getHostelName());
      System.out.println("Room Number       : "+getRoomNumber());
      System.out.println("Room Type    : "+getRoomType());
      }
```

```
}
```

Step 4: Use the file **DayScholar.java** file created earlier. In the ***displayStudentInformation()***, invoke the getter methods of the ***Student*** class to display the values of the instance variables of the ***Student***. This is followed by the display of details of the ***DayScholar*** class members

Step 6: Compile the **Student.java, Hostelite.java, DayScholar.java** and **DemoOverriding.java** (created in Day 4) and execute the code. Verify the output. Which kind of polymorphism is discussed here? What is the use of the ***displayStudentInformation()*** in the ***Student*** class ?

Estimated time: 20 minutes

Summary of this assignment:
In this assignment, you have learnt

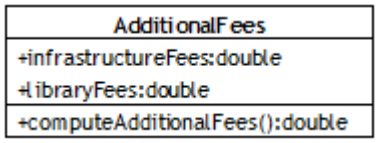
- Abstract keyword and Overriding of methods

Assignment 2: Interfaces

Objective: Understand interfaces

Problem Description: Interfaces represent a special form of classes containing only abstract methods. They are deployed when some generic behavior is required across many classes (both related and unrelated). The student has to pay infrastructure and library fees along with yearly fees. Library and Infrastructure fees are constant values. Implement the same using an interface.

Create an interface called ***AdditionalFees*** as shown below:



Step 1: Create a file called **AdditionalFees.java** and create an interface ***AdditionalFees*** as shown. Initialize the infrastructure fees to 20000.0 and library fees to 6000.0.

Step 2: Compile the file. Do you notice an ***AdditionalFees.class*** getting created?

CONFIDENTIAL

Step 3: Use the **Registration.java** created in Assignment 7 of Day 4. Use the implements keyword so that the ***Registration*** class implements the ***AdditionalFees*** interface and overrides the ***computeAddtionalFees()*** as shown below:

```
class Registration implements AdditionalFees{
…….
public double computeAdditionalFees(){
            return infrastructureFees+libraryFees;
      }
}
```

Step 3: Compile **Registration.java**

Step 4: Can you instantiate an object of an interface. If not, Why?

Estimated time: 20 minutes

Summary of this assignment:
In this assignment, you have learnt

- Interfaces

Assignment 3: Course Registration - the complete program

Problem Description: The parameters to calculate the fees paid by a student are the following:

- Qualifying marks
- Hostelite or Day scholar
- Year of Engineering
- Additional fees to be paid

The complete program has to be integrated using all the classes written.

The first two parameters have been covered in the ***calculateFees()*** written in Assignment 7 of Day 4. Let us cover the last two parameters and calculate the fees for a student.

Step 1: Use the **Registration.java**, **Student.java**, **Hostelite.java**, **DayScholar.java, AdditionalFees.java**, **Address.java** and **PrintDetails.java** created earlier

Step 2: In the **Registration.java** file, consider the ***calculateFees()***:

CONFIDENTIAL

a. Recall the following validations which were already covered in ***calculateFees()*** method, using the ***Student*** reference passed to it:

   i. Invoke the ***validateStudentName()*** method. If the name is valid, invoke the ***validateBranchName()*** and ***validateExamMarks()*** method. If both are valid, go to the next step. Otherwise display appropriate error messages and return 0.

   ii. Based on the branch of engineering, the fees was identified and the discount was given based on qualifying marks. Add one more validation to this by giving discount based on qualifying marks only for first year students (Use the ***Student*** reference variable and use the value of ***yearOfEngineering*** instance variable).

   iii. If the student is a Hostelite, the fees for the hostel is 40000. Calculate the total ***fees(instance variable of Registration class)*** for ***Hostelite*** as follows:

   Total fees = (fees +hostel fees+additional fees)
   Fees to be paid=Total fees - (Total fees * (discount/100))
   Note: Additional fees is the infrastructure and library fees

   iv. If the student is a Day Scholar, then the fees for the bus needs to be paid based on the distance as shown in the table below:

   Table 18

distance	Bus fees
1 – 5	300
6 – 10	500
>10	700

   Calculate the total ***fees(instance variable of Registration class)*** for ***DayScholar*** as follows:

   Total fees = (fees + bus fees+ additional fees)
   Fees to be paid=Total fees - (Total fees * (discount/100))

A sample skeleton is given for your reference

```
class Registration implements AdditionalFees{
……….
public void calculateFees(Student stu){
……
      if(stu.getResidentialStatus()=='H'){
```


CONFIDENTIAL

```
            totalFees = fees +40000+ computeAdditionalFees();
            fees=totalFees-(totalFees*(discount/100))
      }
      else if(stu.getResidentialStatus()=='D'){
            DayScholar day=(DayScholar)stu;
            if(day.getDistance()>=1 && day.getDistance()<=5){
                        busFees=300;
            }
            else if (day.getDistance()>=6 && day.getDistance()<=10){
                        busFees=500;
            }
            else if(day.getDistance()>10){
                        busFees=700;
            }
            totalFees = fees +busFees+ computeAdditionalFees();
            fees=totalFees-(totalFees*(discount/100));
            }
            stu.displayStudentInformation();
}
……..
}
}
```

Notice the typecasting back to DayScholar type as the stu reference would not be able to access the specific methods of the DayScholar class

Notice the calculation of additional fees by invoking the overridden method

v. Invoke the ***displayStudentInformation()*** method using the reference of student

Step 3: Create a file **DemoCourseRegistration.java** with the class ***DemoCourseRegistration*** which contains the main method. Do the following in the main method:

a. Create a reference variable of ***Student*** class called ***studentHostelite*** and instantiate the same to an object of ***Hostelite*** class with the following values (Use the parameterized constructor of respective class):

Table 19

Instance Variable	Value to be initialized
hostelName	Lilliput
roomNumber	109
roomType	single
studentName	Jerry
qualifyingExamMarks	75.6f

residentialStatus	H
branchName	MECH
yearOfEngg	1
addressLine	L-52,SriLabdhi Colony
city	Chennai
state	TamilNadu
zip	600004

(Hint: Create reference variables of Address class whenever an address needs to be stored)

b. Create a reference variable of Registration class called ***registrationStudentOne***.
c. Using the ***registrationStudentOne*** object of ***Registration*** class invoke ***calculateFees()*** by passing the ***studentHostelite*** as an argument. (This is done to register the student to the first year of engineering).

g. Invoke the corresponding getter method to display the fees as shown below:

Fees to be paid by Hostelite for year <Year of engineering>
:_____

Hint:Replace <Year of engineering>with the actual value of the variable>

d. Invoke the ***setYearOfEngg()*** of the Student class using the ***studentHostelite*** object and pass 2 to it.(This is done to register the student to the second year of engineering)
e. Create a reference variable of Registration class called ***registrationStudentTwo***.
f. Using the ***registrationStudentTwo*** object of ***Registration*** class invoke ***calculateFees()*** by passing the ***studentHostelite*** as an argument. (This is done to register the student to the first year of engineering).

h. Invoke the corresponding getter method to display the fees as shown below:

Fees to be paid by Hostelite for year <Year of engineering>
:_____

Step 5: Compile the **Registration.java,** and **DemoCourseRegistration.java** files, execute the program and verify the output

Note: You can repeat Step 3 on a DayScholar object and verify the output

CONFIDENTIAL

Estimated time: 50 minutes

Summary of this assignment:
In this assignment, you have learnt

- Abstract classes ,Dynamic polymorphism and typecasting of objects

Assignment 4: Packages

Objective: Understand the usage of package

Problem Description: Create, compile and execute the program contained in a package and access the data members with different access specifiers in different packages

Step 1: Write the following program in a TextPad

```
package pack1;
public class Base{
      private int privateNum;
      int defaultNum;
      protected int protectedNum;
      public int publicNum;
      public Base(){
          privateNum=90;
          defaultNum=900;
          protectedNum=9000;
          publicNum=90000;
      }
}
```

Step 2: Save the program as Base.java
Step 3: Go to **Configure** menu- Select **Preferences…**option
Step 4: In the Preference window, Click on **Tools** - Select **Compile Java**
Step 5: Tick on **Prompt for parameters** – Click on **Apply** button – Click on **OK**
Step 6: In the Preference window, Click on **Tools**- Select **Run Java Application**
Step 7: Tick on Prompt for parameters – Click on **Apply** button – Click on **OK**
Step 8: Compile the Base.java by typing the following command in the Parameter box
-d . Base.java
Note: Settings given in Step3 to Step7 need to be done only once

Note how one package can be imported into another package

Step 9: Write the following program in TextPad

```
package pack2;
import pack1.Base;
 class Derived extends Base{
     public void disp(){
         // System.out.println(privateNum);
          //System.out.println(defaultNum);
            System.out.println(protectedNum);
            System.out.println(publicNum);
     }
}
class Demo{
   public static void main(String args[]){
       Derived obj=new Derived();
       obj.disp();
    }
}
```

Private and default data members cannot be accessed outside the package

Note how the protected and public data members are accessed in the derived class specified in another package

Step 10: Save the program as **Demo.java**

Step 11: Compile **Demo.java** by typing the following command in the Parameter box

-d . Demo.java

Step12: Execute the program by typing the following command in the Parameter box

pack2/Demo

and observe the output

Note: Refer to the file, "Compilation and Execution of a Java program in a Package.pdf" for understanding the creation of packages

Estimated time: 20 minutes

Summary of this assignment:

In this assignment, you have learnt

- How to create packages, import classes from one package into another

Assignment 5: Packages for Course Registration System

Objective: Create packages for proper organization of classes

Problem Description: Organize the classes created for the case study into a package.

Step 1: Open the files **Student.java**, **Address.java, Hostelite.java**, **DayScholar.java**, **Registration.java**, **PrintDetails.java**, **AdditionalFees.java** and add the following statement as the first line in the file and make all the interfaces and classes as public :

```
package courseRegistration;
```

Step 2: Compile the files individually using the –d option as shown in Assignment 4 of Day 5.

Step 3: Open the **DemoCourseRegistration.java** file, add the following statement as the first line in the file:

```
package Main;
import courseRegistration.*;
```

Step 4: Compile the **DemoCourseRegistration.**java and execute the program and verify the output.

Estimated time: 25 minutes

Summary of this assignment:

In this assignment, you have learnt

- Packages

Assignment 6: Course Registration System - Observations

1. How many lines of code is written for the entire system?
2. How many classes have been identified?
3. How have you addressed the complexity of the code?
4. What are the advantages of following OO approach?

Self Review

What would be the output of the following programs? If there are errors, debug the programs.

Also mention the inference drawn from each snippet provided in a notepad.

1.

CONFIDENTIAL

```
abstract class Animal {
    abstract static void testClassMethod();

    public void testInstanceMethod() {
        System.out.println("The instance method in Animal.");
    }
}

class Cat extends Animal {
    public static void testClassMethod() {
        System.out.println("The class method in Cat.");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Cat.");
    }

    public static void main(String[] args) {
        Cat myCat = new Cat();
        Animal myAnimal = myCat;
        Animal.testClassMethod();
        myAnimal.testInstanceMethod();
    }
}
```

2.

```
interface Count {
    short counter=0;
    void countUp();
}
public class  TestCount implements Count{
 public static void main(String args[]){
       TestCount test=new TestCount();
        test.countUp();

}
public void countUp(){
      for (int x=6;x>counter;x--,++counter){
            System.out.println(" "+counter);
      }
}
}
```

3.

```
interface ParentOne{
      int pOne=1;
      void printParentOne();
}
```

```
interface ParentTwo{
      int pTwo=2;
      void printParentTwo();
}
interface Child extends ParentOne, ParentTwo{
      int child=3;
      void printChild();
}
class InheritClass implements Child{
       public void printParentOne(){
             System.out.println(pOne);
     }
        public void printParentTwo(){
             System.out.println(pTwo);
       }
     public void printChild(){
                  System.out.println(child);
     }
}
class TestInterface{
      public static void main(String args[]){
            InheritClass  inclass=new InheritClass();
      inclass.printParentOne();
      inclass.printParentTwo();
      inclass.printChild();
     }
}
```

4.

```
class Example{
            public final int VAR;
            public void disp(){
                        System.out.println("disp in Example");
            }
}
class Example1 extends Example{
            public void disp(){
                        System.out.println("disp in Example1");
                        System.out.println(VAR);
            }
}
class Demo{
            public static void main(String args[]){
                        Example1 obj=new Example1();
                        obj.disp();
```

```
            }
}
```

5.

```
class Example{
            public final int VAR;
            public void disp(){
                        System.out.println("disp in Example");
            }
}
class Example1 extends Example{
            public final void disp(){
                        System.out.println("disp in Example1");
                        System.out.println(VAR);
            }
}
class Demo{
            public static void main(String args[]){
                        Example1 obj=new Example1();
                        obj.disp();
            }
}
```

6.

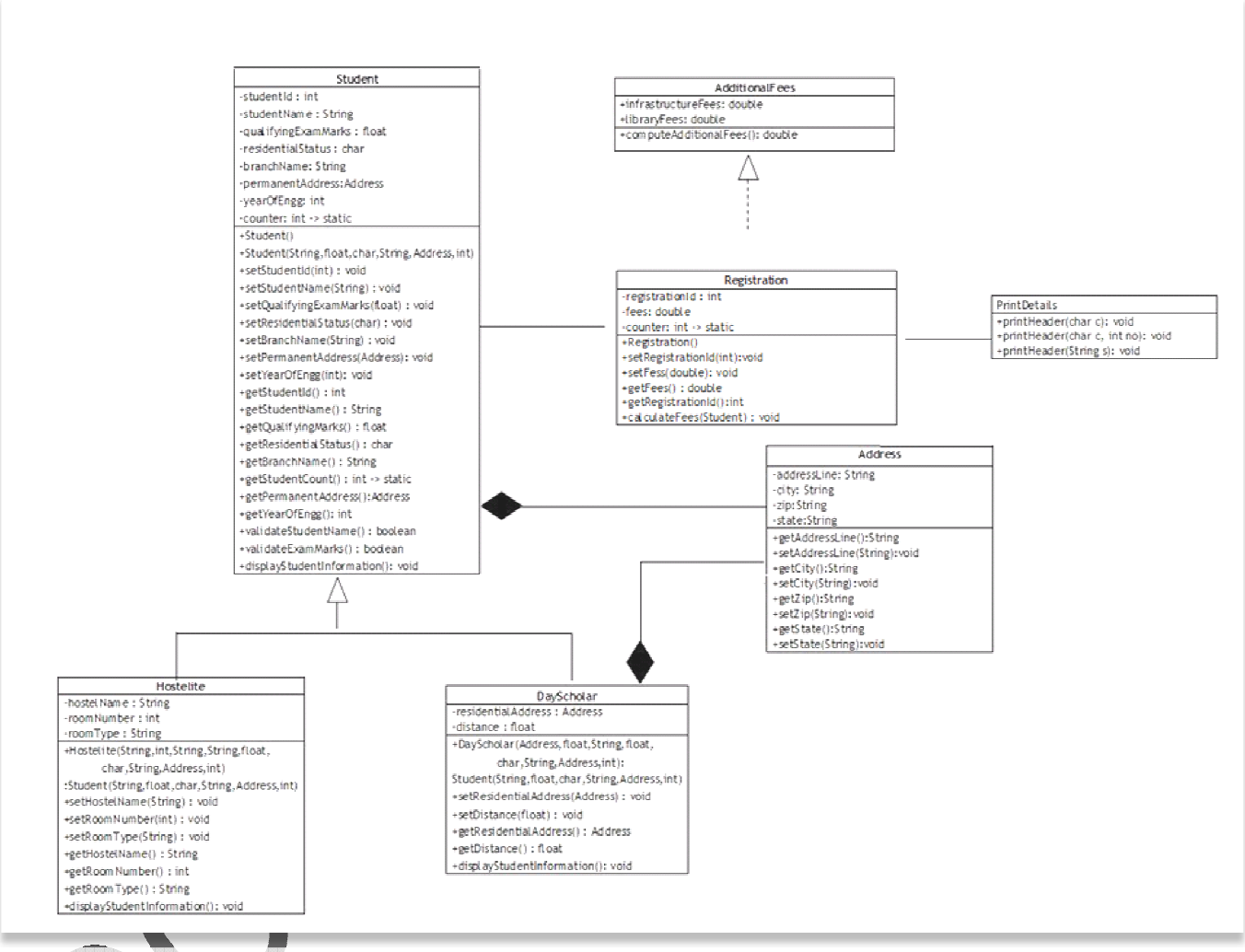
```
package pack1;
public class Base{
     private int privateNum;
     int defaultNum;
     protected int protectedNum;
     public int publicNum;
     public Base(){
         privateNum=90;
         defaultNum=900;
         protectedNum=9000;
         publicNum=90000;
     }
}

package pack3;
import pack1.Base;
```

```
class Sample{
    public void disp(){
       Base obj1=new Base();
       System.out.println(obj1.privateNum);
       System.out.println(obj1.defaultNum);
       System.out.println(obj1.protectedNum);
       System.out.println(obj1.publicNum);
   }
}
class Demo1{
   public static void main(String args[]){
       Sample obj=new Sample();
       obj.disp();
    }
}
```

CONFIDENTIAL

At the end of Day 5…..

The class diagram implemented so far….



At the end of Day 5 you have the following files

Folder name	File name(s)
Assignment 1	DemoOverriding.java, Student.java, Hostelite.java, DayScholar.java, Address.java
Assignment 2	AdditionalFees.java, Registration.java, PrintDetails.java
Assignment 3	DemoCourseRegistration.java, Student.java, Hostelite.java, DayScholar.java, Address.java, Registration.java, PrintDetails.java, AdditionalFees.java
Assignment 4	Demo.java, Base.java
Assignment 5	DemoCourseRegistration.java, Student.java, Hostelite.java, DayScholar.java, Address.java, Registration.java, PrintDetails.java, AdditionalFees.java

CONFIDENTIAL