#### **Ref/Def Cwk: Household Installation Service Simulations – HISS**

1. **The Assignment - see: RefDef Cwk HISS - Case Study**

Produce an **Netbeans** project implementing the specified HISS interface by completing tasks described below**. Your project must compile and be capable of running correctly using Netbeans in our labs.**

Your HISS project should have qualities associated with good program design. It should:

* minimise code duplication and be modularised so modules have low coupling and high cohesion.
* use inheritance, abstract classes and interfaces, as appropriate.
* be reusable and easy to maintain.
* be well documented, displaying both agreed standards of internal documentation and good use of the facilities available in Javadoc.

**If your project does not compile, you will not be able to perform in the demo**. You must attend the demonstration test. **If you do not attend the demonstration test, you will get ZERO marks**

The project hiss-studentsNB contains the interface HISS, and a number of other classes.

Except as instructed, **you must NOT change interface HISS** or the **Manager.java** constructors. IF you do, you will be **penalised**.You may amend or add other classes/methods to ensure functionality.

1. **Paired Programming**

You may do this coursework:

* **EITHER** as an **INDIVIDUAL** – **add your name and SRN in the Teamwork class**
* **OR** as one of a **PAIR** – **add your name/ SRN and the name/SRN of your partner in the header comments of Manager.** NOTE:- you must EACH submit the project to Studynet, so that you can each do the demo (see below)

**Note: You may only pair with another student who has been either Referred or Deferred**

For paired programming to be successful, the pair must be able to work together. You should also vary who actually types in the code and who is observing/advising. This technique works best if the skill levels of the pair are roughly equivalent. If there is a wide gap between skill levels the weaker of the pair will not learn anything, while the stronger will do all of the work. This will not prepare the weaker partner for the tasks in the demo test. **If completing as a PAIR:**

* Edit the Manager header to include names/SRN of both.
* EACH submit the project to Studynet by the deadline and save the project to their UH One Drive.
* Tasks in the **demo** test must be done **individually.**

1. **Assignment Tasks**

**Task 1- Test Driven Design (10 marks)**

In "test first development" you should design the tests before coding, and you then write code to produce the expected results. Complete a test plan for HISS using data from Case Study Appendix A.

*(A possible entry is shown):*

**Test Plan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Purpose** | **Test Data** | **Expected Result** | **Actual result** |
| *1.* | *Hire staff* | *Amir* | *Amir in Team, account = 800* |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Task 2 – Programming**

**Task 2.0 – Getting started**

* Download and saveHISS-studentsNB. Do **NOT** change the code provided, but do add to it.

HISS and the ManagerUI classes, cover the functional requirement. You must NOT change HISS, but should implement its methods in Manager. Do **NOT** change the Manager constructors

* Before you try to implement Manager, return to the specification and identify useful supplier classes which may be helpful in implementing Manager. If you try to just code everything in Manager, the code will be very complex and unmaintainable, and so not get good marks for design. For each class, identify suitable fields and methods. Also, identify opportunities for using inheritance and polymorphism. (Similarity to BATHS is NOT a coincidence !).

**Task 2.1 – Initial setup `**

* First, implement the lower level supplier classes identified in task 2.0. Include some of the fields, at least one constructor, and a toString(). You can always add further methods alter. Consider opportunities for using inheritance.
* Manager has “stub” methods for all methods specified in HISS; they don’t contain any code but accessors have “dummy” return values to allow **Manager** to compile.
* Declare appropriate fields in **Manager** and complete at least the one constructor which requires two parameters: the manager's name and project budget, which can be used to initialise **Manager** fields.
* Decide how to store staff and jobs. You will need to provide lists of staff available for hire, the project team and jobs. Consider using collection/map classes from the Java library to manage this. Declare and create appropriate collection/map variables (fields) in **Manager**.
* Provide two **private** methods **setUpStaff()** and **setUpJobs()** which should actually add the staff and jobs listed in Appendix A to these collections/maps. These two methods must be called in your **Manager** constructor, to “load” data into the **Manager** object. Creating and storing the data shown in Appendix A in appropriate collections should be done only in these methods. (To check whether your code works use **ManagerUI** options).
* Test the loading of data, by usingeither **ManagerUI** or JUnit **GeneralTest**

**Task 2.2 – Producing the basic HISS system (30 marks)**

* Your system should provide the functionality specified in the HISS interface, by providing implementation for the methods in the **Manager** class which currently exist only as “stubs”.
* You MUST NOT change interface **HISS** (except as specified)
* ONLY methods specified in **HISS** interface should be public in **Manager**. However, you can provide additional private methods in **Manager** to improve readability & maintainability, reduce duplication or perform well-defined tasks.
* An incomplete **Manager** class has been provided. You should complete this class
* This task will be tested automatically by using JUnits
* A large portion of marks is for the correct implementation of **hireStaff()** and **doJob** **()**

**Task 2.3 – Design (10 marks)**

* Your design should include appropriate supplier classes to handle the detailed implementation of the system and include some appropriate robustness checks
* Marks will be awarded for appropriate: classes, use of inheritance/polymorphism, devolving functionality to lower level classes, private methods to improve readability/maintainability, low coupling, high cohesion
* Classes should not provide output directly to the Terminal window (only **ManagerUI** and **ManagerGUI** should output to terminal).
* Two **enum** classes **StaffState and JobType** have been provided. You are not required to use these classes, but not using them may cause delays in completing the coding.
* There are design marks for appropriate use of inheritance and enums in the system. You should also provide **private** methods in **Manager** to improve readability, reduce duplication or perform well-defined tasks

**Task 2.4 – Documentation & Testing (12 marks)**

* Your code should show appropriate use of naming conventions, layout and Javadoc comment
* Project functionality will be marked by running JUnit tests. These testing tools are provided for you
  + **A command line user interface ManagerUI** tests some of the Manager functionality "at the keyboard". Some of the code has been omitted. Provide implementation for options:

4. list staff in the team

6. do a job

Obtain any information that you require from the user, and then call appropriate methods specified in **Manage.**Provide appropriate output for the user on the result of their option choice.

* + **JUnits test suite**

A JUnit test suite for automated regression testing is provided in the test folder. It does not provide a complete test of the functionality. If your code passes these tests, you are assured of some marks, but may miss marks for some features which are not tested in this suite of JUnits.

* Unzip and add the test folder to your Netbeans project
* **Ensure the test suite works** (see: “JUnit Instructions for Netbeans.in Unit :Technical)
* Use the test suites to check your own code
* Extend the JUnit suite be adding your own tests in a section/s clearly labelled \*\*\*My Tests\*\*\* within tests suites. Your tests should be documented by stating what they are testing

**Task 2.5 – Persistence (10 marks)**

* In the Manager class provide methods :
* saveManager() to write the whole Manager object using serialization to a file whose filename is provided by a parameter.
* restoreManager() to read using serialization the whole Manager object from a file whose file name is provided by a parameter .
* a **private** method in Manager to read data about jobs from the text file jobs.txt (provided).
* Provide a second constructor in Manager which in addition to the name and budget also requires a filename as a parameters and calls readJobs()to "load" job information (instead of calling setupJobs()).Staff can still be loaded by calling setupStaff(). When testing this constructor, pass it the filename jobs.txt which has been provided in HISS-students
* This task will be tested using a suite of JUnits

**Task 2.6 – Graphical User Interface – *Challenge Task* (10 marks)**

A partially completed ManagerGUI provides for a GUI for users. Complete this class to provide :

* To the Jobs menu add the following options:
* list the jobs – display the job list in JTextArea listing
* do a job – use suitable JOptionPane objects for input and to output the result
* To the eastPanel, add the following buttons:
* Hire staff – use suitable JOptionPane objects for input and to output the result
* list the team – display the project team list in JTextArea listing
* Clear – clear the contents of JTextArea listing

NOTE:

* This class has just a sample of possible functionality used to demonstrate that you can code a GUI.
* There is NO need to ensure that ManagerGUI provides all of the HISS functionality, just those above
* There are many (and more advanced) facilities for implementing GUIs in Java; for this assignment you are asked to **amend** the ManagerGUI class provided.

1. **Technical Report (11 marks)**

This report should be a word-processed document of no more than **500 - 1000** words (max 2 pages of A4). It should include:

* a UML diagram (BlueJ style) of your project design showing relationships between classes
* THREE important design decisions which you made in implementing this project.

For EACH design decision:

* State your design decision
* Identify possible alternative implementations
* Discuss the pros and cons of your decisions with reference to these alternatives
* Explain the reasons for your final decision

1. **Demonstration Test (30 marks)**

AFTER the assignment hand-in, you must attend a demonstration test to show that you have a good working knowledge of the code that you submitted.

* This is an in-class test which requires your attendance in-person.
* You will be expected to use the computers in our labs (NOT your own laptop).
* You will be given a demo specification on paper to implement
* You must NOT use GenAI to write the required code

The demo specification will ask you to:

* open your submission saved to UH One Drive
* In a timed test of 120 minutes, make the specified changes to the project on your own computer
* upload a zipped amended project to the RefCwk HISS-Demo assignment slot.

The main purpose of the demonstration is to authenticate your code by showing that you know it well enough to use it and make these changes**. If you do not undertake this demonstration, your assignment will get ZERO marks.**

At the demo, you may be asked to:

* write a demo class to test the functionality of your system
* add a specified subclass
* add a new class, or methods to an existing class
* amend Manager to add actual jobs or staff
* Add methods to Manager
* Amend ManagerGUI – IF you want to get credit for task 2.6 above

Marks for the demonstration will be for code to perform specified tasks, not for producing correct output. To see the type of tasks which could be asked, see the mock demo posted for the original BATHS assignment

**Marking & Feedback**

Marks awarded on the feedback sheet total 120 and will then be converted to a %. And capped if necessary (due to demo mark)

Marks for the initially submitted project will be awarded in the following way:

* **Functionality** will be marked automatically for functionality by JUnits using our test data.
* **Design** marks will be awarded on inspection of your code by staff.
* **Additions** may be made for interesting features in your code.
* **Deductions** may be made for breaking some of the principles of good object-oriented design

Both members of a pair will initially be awarded the same mark.

Individual marks will then be limited by your individual performance in the demo.

* Individual marks for the whole assignment will be limited in the following way:

|  |  |
| --- | --- |
| **Demonstration Mark** | **Maximum Coursework Mark** |
| 0<= mark < 3 | 10 |
| 3 <= mark < 8 | 45 |
| 8 <= mark <15 | 60 |
| 15 <= mark <= 30 | 100 |

**Assignment Grading Criteria**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Generic Grading Criteria | no merit | clear fail | marginal fail | marginal pass | satisfactory | good | very good | outstanding |
| Marks | 0 - 19 | 20 - 29 | 30 - 39 | 40 - 49 | 50 - 59 | 60 - 69 | 70 - 79 | 80 - 100 |

**Submission Requirements**

**1. Submit to Studynet by 24th June 2025 by 20.00**

A zipped file containing :

* a **Netbeans** project developed for Tasks 2.1 – 2.6
* A document with your test plan
* A document with your technical report

Your project must compile even if all the tasks have not been fully completed. **You will be penalised if your code does not compile.** Enclose code which does not compile in comments

**2.** **Save** your Netbeans project to your UH One Drive

**3. Participate in a timed demonstration activity – to be confirmed 25th June**

At this test you will have 120 minutes (SNA 150 ) to complete the following

* download your original submission
* perform the tasks on the demo briefing sheet
* when done, upload your zipped completed demo to RefCwk-Demo assignment slot on Studynet

**// see below for Appendix A**

**Appendix A**

**Staff**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Name** | **Experience** | **Retainer** | **Corgi** | **Trained** | **Make** | **HourlyRate** |
| Planner | **Amir** | **2** | 300 |  |  | Homebase | 2\*15=30 |
| Consultant | **Bela** | **2** | **100** | **false** |  |  | 30 |
| Consultant | **Ceri** | **4** | **250** | **true** |  |  | 40 |
| Installer | **Dana** | **2** | 200 |  | no |  | 20 |
| Installer | **Eli** | **7** | 200 |  | yes |  | 20 |
| Planner | **Firat** | **6** | 300 |  |  | Hygena | 6\*15 = 90 |
| Installer | **Gani** | **2** | 200 |  | yes |  | 20 |
| Consultant | **Hui** | **8** | **450** | **true** |  |  | 40 |
| Planner | **Jaga** | **4** | 300 |  |  | Homebase | 4 \* 15 = 60 |

**Tasks**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Task No** | **Type** | **Location** | **Experience Required** | **Hours** | **Penalty** |
| 1 | Design | Kitchen | 3 | 10 | 200 |
| 2 | Maintenance | Lounge | 3 | 20 | 150 |
| 3 | Installation | Kitchen | 3 | 30 | 100 |
| 4 | Design | Bathroom | 9 | 25 | 250 |
| 5 | Installation | Lounge | 7 | 15 | 350 |
| 6 | Maintenance | Kitchen | 8 | 35 | 300 |
| 7 | Maintenance | Bathroom | 5 | 20 | 400 |
| 8 | Installation | Bathroom | 1 | 5 | 120 |
| 9 | Design | Kitchen | 1 | 8 | 175 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Design** | **Installation** | **Maintenance** |
| **Planner** | yes | no | no |
| **Installer** | no | yes | Only, if trained |
| **Consultant** | yes | yes | yes |