**5COM2007** **Cwk 3: Zenith Attractions and Pleasure Parks(ZAPP)**

**Background**

The ZAPP organisation manages some pleasure parks offering leisure activities. These parks always consist of areas with different leisure facilities. Each area is connected by bridges to some of the other areas at the park. Guests at a park are given a personalised card which must be carried when moving around the park. Cards can be loaded with credits. A card must be used to make a bridge crossing from one area to another area and each journey costs a number of credits. In addition, there are some conditions to be met before anyone can use a card for a bridge crossing. Credits for cards can be bought at any time. This project is concerned with managing the movement of guests between areas at a park using bridges and will be implemented at the Fantasia Pleasure Park (then rolled out to other parks).

**System Requirements**

1. The following basic **functional** requirements have been established for the proposed system:

* Display details of the whole park
* List all cards in all areas of the park
* List all cards currently in a specified area
* Find the current location of a card
* Authorise a bridge crossing, if it meets the required conditions
* View data on a card
* Manage a card’s credits , including convert points to credits

1. The following **data** requirements have been established:

* **Card** – used by people at the park for location and moves.

Information stored about a card includes: a card ID number, guest name, a luxury rating, a number of credits and a number of points

The Card class constructor uses parameters to set the card id, the name, the luxury rating, the number of credits and sets the points to 0. The luxury rating (1 to 10) determines the areas which the person is allowed to visit e.g. a card with a luxury rating of 3 is allowed to visit to all areas rated 3 or below.

When a card is used to make a move over a bridge a basic 4 credits is deducted and 1 point added as people use the bridge. 3 points can be converted into 1 credit. People may top up their credits or convert points to credits at any time. (Handling the actual payments is outside the project scope)

* **Area** – a park consists of a network of areas

Each area has a reference number, a name, a rating and a capacity (the maximum number of people/cards that can be in the area at any one time). Each area should maintain a list of all cards currently in that area. These lists should be updated whenever a card enters or leaves an area, so that it is always possible to say who is currently in an area and to find the location of a card.

* **Bridge** – connects two areas at the park

Each bridge has a code and connects a source area to a destination area. It represents a crossing in one direction only. To make a move over a bridge, certain conditions must be met and these are enforced by card readers at the entrance to the bridge If these conditions are met, the system allows the person onto the bridge to move to the destination area. It updates its records to show that the card has left the source area and has travelled to the destination area. It also updates the card's credits and other information, as appropriate.

A request by someone(card) to move over a bridge will produce one of the following results:

* Refusal to cross the bridge, because :
* the card's luxury rating is lower than the rating of the destination area.
* the arrival of the card would exceed the maximum capacity of the destination area.
* the card does not have enough credits for the crossing
* the card is not listed in the source area for the bridge
* Successful entry, because none of the above conditions is true

In addition, the system should refuse requests for non-existent cards or bridges

* **Park - implements the ZAPP interface**

A park has a network of **areas** joined by **one-way bridges**. It also has a collection of **cards**

A park must always have a “Lobby” area which stores all cards. Its name MUST be "Lobby" and it MUST have a rating of 0 and a large maximum capacity. All cards belonging to guests using a park are initially added to this “Lobby” area.

A card arriving or leaving the whole park is also outside the scope of this assignment.

**Paired Programming**

* The main part of this assignment may be completed by working in a **pair, or individually**.
* The **demo** part of this assignment (Task 8) must be completed **individually**
* You must complete information required in the Teamwork class, even if you work on your own
* For paired programming to be successful, members must be able to discuss and work together, for suitably long periods of time. You must, therefore, identify time slots in which you can both work together. During your session together, you should vary who types in the code and who is observing/advising to ensure an even distribution of work. Screen sharing is a useful tool.
* Paired programming works best if the skill levels of the pair are roughly equivalent. If there is a wide disparity between skill levels, there is a high probability that 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 future assessments.
* Partners do not have to attend the same practical, as practical sessions should NOT be used to complete this assignment (but you may use them to get to get help).
* Partners must EACH submit the same project file to Studynet by the deadline

**The Assignment**

A BlueJ project zapp-students is provided. Amend this BlueJ project so that it implement a version of the ZAPP system by completing tasks described below. .

* Your ZAPP project should display the qualities associated with good program design:
* Your system should minimise code duplication and be modularised so that components have low coupling and high cohesion.
* You should aim to make your code reusable and easy to maintain.
* Program code should be well documented, displaying agreed standards of internal documentation.
* Lower level classes should not contain input/output statements
* Marks shown for each task include marks for both functionality and design.
* You will also be required to provide an IntelliJ version of your project
* **Your project must compile**. Code which does not compile may be included as comments
* You must attend the **online timed demo session**. **If you do not attend, you will get ZERO marks.**

**Assignment Tasks - Marks awarded for tasks below total 120 and will be scaled to %**

**Task 1 - Implement and test the Card class 12 marks**

* **class Card** should have, at least:
* the fields described above
* a constructor to set all fields values, either using parameter values, or default values (see above)
* accessors to return the card id, the luxury rating, number of credits
* methods to add or deduct credits
* a method returning boolean true, if the card has enough credits for a bridge crossing
* a method to update credits and points when a card uses a bridge
* a method to convert points to credits
* a toString() method returning a String with all relevant information about the Card

You may add further methods, if you consider them useful.

* **class CardTester -** which can be run to show that the Card class works properly. It should:
* create some suitable Card objects
* and call their methods
* show that actual results are as expected

**Task 2 - Implementing other lower level classes 18 marks**

* **class Area** should include:
* suitable fields as specified above.
* a constructor to create an area using suitable parameters
* accessors to return its area number, name, luxury rating
* an ArrayList field to store Card object references. (declared and created)
* mutator ; enter()which has a Card as a parameter and adds it from the ArrayList
* mutator leave()which has a Card as a parameter, finds its position in the ArrayList and then removes the card using its position.
* an accessor which either says whether the area is full (reached capacity) (..or still has capacity)
* method to list all the Cards currently in the area
* methods to find and return details of one Card in the area
* an accessor which returns a boolean saying whether a Card is in the area (in the ArrayList)
* a toString () method which includes area details and a list of the Cards in the area

You may add more methods if you require them. You should NOT have mutators which change the basic information about an area ( reference number, name, luxury rating) as these should not change

* **class** **Bridge** should include:
* suitable fields as specified above. The "from" area and the "to" area should be Area objects
* a small range of accessors/mutators to process data held by the class
* method which returns a boolean saying whether a card given as a parameter can enter the bridge
* a method to process a card moving to the destination area;
  + if the card meets the conditions, remove it from the source area, add it to the destination area, update information on the card, and **return** a suitable message
  + if it does not meet one of the conditions, **return** an appropriate message and do NOT move it
* a method toString()which returns a String representation of an object of that class (no need to list all of the area details, just the area numbers and names)

You may add further fields/methods to these classes if you require them during further development

**IMPORTANT NOTE: these lower level classes should NOT be using System.out.println()or requiring keyboard input from a user**

**Task 3 - Implementing the Park class 18 marks**

* **interface** **ZAPP**

This class specifies methods required to provide system functionality. A fully documented version of **ZAPP** is available in zapp-students and you have **NOTHING FURTHER** to do in this class.

**You must NOT change any signatures of methods in the ZAPP interface – else you will be penalised**

* **class Park**
* a framework for this class is included in zapp-students
* is set up to implement the **interface ZAPP,** so do NOT change its header
* has been produced by copying the methods specified in the interface, and providing code "stubs" where you can write your code.

You should use this framework to:

* declare and create three collections to store references to: all areas, all bridges & all cardes.
* the collection of all areas should be implemented as an Arraylist, and the positions of the areas in the ArrayList is the same their numbers i.e. Area 0 Lobby should be in location 0
* the collections of cardes & bridges may also be Arraylist, but there are no assumptions about the order of objects
* provide code for a constructor which:
* sets the name of the park from a parameter (see constructor header)
* it should then call three private methods, to be defined at the end of the **Park** class
* **loadAreas** - which creates all the areas in **Appendix A** **below**,and adds them to the collection of areas. These should be added in the order of their id numbers.
* **setUpBridges** - which creates all bridges carding area objects to their constructors, thren adding them to the collection of bridges.
* **loadCardes** - which creates all cardes and add them to the collection of cardes
* finally, the constructor should add all of the cardes to the “Lobby”area
* the class should then provide implementation for all methods specified in the **interface** **ZAPP,** as well as the three private methods, outlined in **Park** above
* **NOTE: move()** and  **canMove()** carry the majority of marks in this task. They should return an appropriate message for the outcome. They should include a message if either the card or the bridge are NOT part of the park.
* any additional methods which you may/should want to add to improve the design (but which are not specified in the **interface**) should be declared as **private**
* **Helpful methods**

[3 private methods at the end of this class, to find and return each type of object (bridge, card, area) from its collection using key data, may be useful, but not required]

**Task 4 - Implementing an application 7 marks**

* **class ParkUI** - this is the class which provides an application for the user
* only **ParkUI** and the **MyTester** in Task 6 should use System.out.println
* this is the only class which has input and output from/to the user
* some of the code has been written for you
* you should provide the remaining code as shown by the comments
* this class should only call methods specified in **ZAPP**
* it should compile (but not work) even if you have not yet written the implementation for **Park**

**Task 5 - System Documentation 5 marks**

You should produce:

* a visually neat and readable UML-style class diagram of your system displayed in BlueJ
* program code which is well documented, displaying agreed standards of internal documentation and naming conventions. No need for detailed comments on method code (Park is well documented)
* the MyTester class in Task 6 below should have comments which explain the purpose of each test and expected results

**Task 6 - System Testing 16 marks**

You can test your code by running the ParkUI but you will eventually find this tedious. So, we have provided a skeleton MyTester class to test your Park class. MyTester:

* already declares an object of the Park class named fantasia
* already has a main() to make it runnable (see ParkUI main()) -no need to do anything here
* has a doTest() method in which to **write code to call methods** on the ZAPP variable in a way which tests your system and demonstrates it works according to specification. You should include appropriate output to the terminal window. You must also use comments to explain what is being tested.
* You are NOT required to write any other testers classes except MyTester and CardTester,
* Marks for this task will be awarded for:
* the appropriate choice of data,
* the sequencing of method calls
* appropriate output
* explanations of tests & expected results.

We are looking for evidence of a systematic approach to testing and will expect you to show that you have identified and tested for the main events likely to occur when the system is running. At this stage, you may ignore the situation where a Card moves to an area but has insufficient credits to return to the "Lobby".

**Task 7 – Inheritance – Challenge Task 10 marks**

**You will only get marks for this task, if you complete an extra task in the Demonstration Test (see below) to an acceptable standard**

Provide the following subclasses which extend the Card class.

All should have a toString() method which overrides toString() in Card to include subclass data

* **Tourist card:**
* is created with a specified luxury rating and a number of credits determined by parameter values
* tour operator’s name and operator id (numeric) needs to be set by parameter values
* **Child card**
* are created with the highest luxury rating of 10.Children can visit all zones for 0 credits
* include the parent card number set by parameter value , and number of zones visited set to 0
* deduct 0 credits for bridge crossing, but add 1 to the visit score whenever a bridge crossing is made
* **Company card**
* is created 30 credits, with a luxury rating specified by a parameter value
* the organisation’s name needs to be set by parameter values
* deduct only 3 credits and add 2 points whenever a bridge crossing is made

Now add an object of each class (using your own data) to the Cards in the Park method loadCards()and check the everything compiles and works for these objects

**Task 8 – Demonstration test 24 marks**

After the assignment hand-in, you will be asked to demonstrate that you have a good working knowledge of the code that you submitted. You will be handed a test specification

In a timed session of 100 minutes (SNA 125 mins), you will be asked to

* download original project and rename (as instructed in the Demo specification)
* make the specified changes to the project
* then upload their zipped re-named amended project to the Cwk3 - 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.**

You may be asked to:

* add a new specified class
* add a specified class which uses the Card class
* write a demo class to test the functionality of your system
* amend Park to create a park with small changes to the Fantasia Park
* add a method to a class

**Task 9 – Convert to IntelliJ 10 marks**

Convert your BlueJ project into an IntelliJ project

**Submission Requirements**

If you have completed this assignment as part of a pair, EACH member of the pair should submit the same file, but the complete the demonstration individually.

1. **By Thurs 11th Jan by 20.00**

* Complete the Teamwork class with details of your pair (or just yourself)
* Zip the file containing the BlueJ project developed for Tasks 1 – 7
* Zip the file containing your IntelliJ project converted from the BlueJ project
* Submit to Studynet by the deadline:
  + A zipped BlueJ project
  + A zipped IntelliJ project

NOTE: Your BlueJ project must compile even if you have not fully completed all the tasks. **You will be penalised if you submit code that does not compile.** You may include code which does not compile, but only as comments enclose in comment markers.

**2. Attend the demonstration test session (individually) – date to be announced**

You will be asked to access your submission and perform the activities described on a "Demonstration" briefing sheet given at the session. When you have done this, you will also be asked to upload your completed demo to Studynet and to hand the demo briefing sheet which should include your name and student registration number (and those of your pair, if applicable) back to the supervisor.

**If you do not submit your code to Studynet, you will be unable to do the demonstration**

**Marking & Feedback**

Marks awarded on the feedback sheet total 120. and will then be converted to a %.

The submitted assignment will be marked using a combination of automated testing and visual inspection of code which will form the basis for individual marks. So it is important NOT to change method signatures in ZAPP or Park

Your performance in the demonstration will put a limit on the marks you get for the coursework. In addition, a serious mismatch between your submitted work and your performance in the demonstration may result in proceedings for plagiarism/collusion. The limits on coursework marks are as follows:

|  |  |
| --- | --- |
| **Demonstration Mark** | **Maximum Cwk Mark** |
| mark < 4 | 15 |
| 4 <= mark <= 8 | 45 |
| 8 < mark <15 | 65 |
| mark >=15 | 100 |

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

The %mark will be 25% of the marks for the module

Feedback will be provided in a personalised feedback form posted to Studynet.

**Appendix A: Fantasia Pleasure Park - Areas & Bridges**

**4.Solitaire**

Rating: 1

Capacity:1

**3.WildWest**

Rating: 5

Capacity :2

GHJ6

EFG5

JKL8

HJK7

**1.Concourse**

Rating: 1

Capacity :100

**2.Waterworld**

Rating: 3

Capacity :10

DEF4

CDE3

BCD2

ABC1

**0.Lobby**

Rating: 0

Capacity:1000

#### **Current cards:**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Name** | **Rating** | **Credits** |
| 1000 | Lynn | 5 | 10 |
| 1001 | May | 3 | 20 |
| 1002 | Nils | 10 | 20 |
| 1003 | Olek | 2 | 12 |
| 1004 | Pan | 3 | **4** |
| 1005 | Quin | 1 | 5 |
| 1006 | Raj | 10 | 6 |
| 1007 | Sol | 7 | 20 |
| 1008 | Tel | 6 | 24 |