School of Science

CPT121 / COSC2135 Programming 1 (OUA)

Assessment 3: Assignment 3

Assessment type: Individual assignment; no group work.

Word limit: N/A

Due date: Sunday ending Week 12

(i.e. 23rd May 2021) 23:59 (AEST)

As this is a major assignment in which you demonstrate your understanding, a university standard late penalty of 10% per each working day applies for up to 5 working days late, unless special consideration has been granted.

Weighting: 25%

Overview

This assessment requires you to apply various concepts and techniques covered in weeks 1–11 of the course, to assemble a programmatic solution for a problem based on a simulated real-world scenario.

An important part of your development as a programmer is the ability to identify and apply appropriate techniques or programming structures when implementing the various aspects of a programmatic solution to a given problem, so in addition to the basic functional requirements of this task, you will also be assessed on your ability to select and utilise appropriate techniques and structures in your solution.

This assessment requires you to design and create a Java program to solve analysed stakeholder (user and supervisor) requirements.

Assessment grading criteria

This assessment will measure your ability to:

- Apply the specified coding and documentation style guide
- Apply basic features of Java and Object-Oriented techniques (covered in weeks 1-12 of the course material) to solve the user and supervisor requirements, demonstrated by producing correct and well-formatted output on provided test data

Course learning outcomes

This assessment is relevant to the following course learning outcomes:

- CLO 1: Solve simple algorithmic computing problems using basic control structures and Object-Oriented Techniques
- CLO 2: Design and implement computer programs based on analysing and modelling requirements
- CLO 3: Identify and apply basic features of an Object-Oriented programming language through the use of standard Java (Java SE) language constructs and APIs
- CLO 4: Identify and apply good programming style based on established standards, practices and coding guidelines
- CLO 5: Devise and apply strategies to test the developed software

RMIT UNIVERSITY

Assessment details

'ChildzPlay' is a small (fictional) toy and play equipment hire company which hires out toys, dress-ups and backyard play equipment. Covid-19 lockdowns and school closures have resulted in parents seeking alternative child entertainment. ChildzPlay has experienced overwhelming demand for their services and they urgently wish to computerise some aspects of their record keeping. The first phase is to develop a prototype system for storing details of all items for hire.

Notes from interviews with ChildzPlay staff and instructions from your supervisor are given below. You should analyse the interviews and instructions to determine the functional requirements for your program. There are four stages in this assignment (*A, B, C* and *D*). Since each stage elaborates on the previous one, you only need to submit one program – the one implementing the highest stage you were able to complete.

For each of the stages *A*, *B, C* and *D*, complete the steps below based on the Double Diamond process:

Discover

'Go wide' in the form of reading and thinking about the supplied stakeholder requirements. Find out what the current situation is and understand user behaviours and business drivers.

Fill in the supplied 'Discover and Define' Word template with a list of different problems that could be solved.

Define

From your understanding of the overall issues/requirements, narrow down to a single problem and turn that into a problem statement. The problem statement defines what you will develop, and you may start some initial coding to start working out if you can create a solution to the problem you defined.

Fill in the supplied 'Discover and Define' Word template with a statement of the single problem you will solve.

Develop

Start coding to create an initial prototype and program logic to address the problem. You may create a few different iterations to get to the best prototype. This is a phase for trying ideas out to see if they work.

Deliver

Pick the best prototype from the 'Develop' phase and create the final version of the code, refining and making it work. The aim is to create a minimum viable product (MVP) to address the problem you have identified and defined. The final result of this process at stage *D* will become your final submission for assessment 3.

How to use the Double Diamond is explained in the course webinar, please ask your instructor if you have any questions.

The assessment is marked out of a total of 25 marks as given in the rubric criteria table at the end of this document.

Coding style/documentation

Your program must follow the coding and documentation style guide given in the file *StyleGuide.pdf*, available on the course Canvas under the *Assignment 3* link.

Stage *A* – Modelling and implementing an Item Management System

An edited summary of client interviews and additional requirements from your supervisor is given below:

Kanchan – ChildzPlay Manager

> *"At ChildzPlay we have several types of item, ranging from toys (including sporting equipment) and dress-ups to play equipment. All of them have an item code (incrementing from a starting number of 100 for the first item), a title, a single line description and a cost per week. Obviously I need to be able to add items to the system. For this stage, I'd like the system to be able to store up to five items, so we can easily test it. My book keeper will need to be able to process a customer hiring or returning an item. For this prototype we won't computerise customer records, but be aware that each customer is identified with a (string) ID code. Both of us need to be able to display all information stored about a specified item (including if it has been hired), or to list a summary of item ID, item title and hire status of all items together with a final count of the number of hired items and the total income from all the hired items. All displays should be neatly formatted. "*

Nathan – ChildzPlay Book Keeper

> *"When a customer wishes to hire an item, I collect the customer's ID, and find out which item they wish to hire (the item has a tag or mark on it giving the item code), and for how many weeks they wish to hire it. I then record that the item has been hired by that customer for that many weeks. I then charge the customer and give them a receipt listing their customer ID, the item code and title, the number of weeks, and the total charge (hiring is on a per week basis). As the system is being computerised, I'd like the system to let me know if hiring isn't possible because the item is already hired out (there is no wait-listing).*
>
> *When a customer returns an item, I look up the record for that item code and mark the item as being available for hire."*

Your supervisor advises that your program should be able to use object-oriented techniques to model an item in the class *Item*. The *Item* class shouldn't collect data from the user. Another class, *StageA*, should be a console-driven class that houses the main method, obtains relevant item details from the manager as required, creates and manages objects of class *Item* in order to implement the functionality from the above interviews. The *Item* class must contain a static variable indicating the value of the next id and use this when determining the id for a new item object. The *Item* class constructor should accept user data from *StageA* and set the item attributes accordingly.

In stage A, your program **MUST** utilise a **single** array (called *holdings*) for storing information about all the items, in class *StageA*. Collections such as ArrayLists must not be used. Not meeting this requirement will result in a mark of 0.

Your *Item* class must include and use a Boolean method with signature *hireItem(String customerID, int numWeeks)* which sets item instance variables to indicate the item is being hired out by *customerID* and hired for *numWeeks* of weeks. It returns false if the item is already hired out, true otherwise. Similarly, *Item* must implement a Boolean method called *returnItem()* to return an item. This method returns false if the item wasn't hired and true if it had been hired. *Your* StageA should be designed to use the return values of the *Item hireItem* and *returnItem* methods for hiring/returning items and determining if an attempt is being

made to hire an already hired out item or return an item that wasn't hired out.

The *StageA* class should provide a user-friendly menu to allow the user to perform tasks relevant to the needs expressed in interviews, above, calling methods in *Item* objects as relevant. Your program should be able to handle conditions such as searching for items that don't exist, attempting to add more items than the (user specified) maximum number of items, a customer attempting to hire an item that another customer has already hired etc.

You are expected to adhere to all relevant object-oriented programming guidelines, including:

- Visibility of instance variables and methods set appropriately
- No unnecessary accessors (setters) or mutators (getters) - only provide methods which will be needed when implementing the application class in this stage
- Where object instance variable values can be reasonably determined before object creation, initialisation of these variables should be carried out via a constructor.
- Methods should work with instance variables when performing their required task
- Parameter lists in methods should be appropriate to the task the method is performing - only accept parameters where a method requires one or more values from the caller to perform its assigned task that it does not already have access to
- Methods which need to communicate a value or result back to the caller should do by returning the value in question, not by storing it in an instance variable or printing it to the screen
- Data classes (i.e. Item) should not prompt for input from the user

Stage *B* – Extending the Item class

In this stage you rename class *StageA* as *StageB*, modify and extend the *Item* class and modify the *StageB* class to meet the requirements presented in the interviews and supervisor requirements below.

Kanchan – ChildzPlay Manager

> *"We have several types of item, and don't actually have 'generic' items. We only have toys, dress-ups and play equipment. When adding an item I need the system to ask which type of item is being added and collect details relevant to that item. It shouldn't be possible to add a 'generic' item as in stage A.*
>
> *There are three categories of toy: "construction" (e.g. Duplo™), "ride-on" (e.g. balance bikes) and "sport" (e.g. basketball hoop and ball). There are fixed costs for each of these: construction equipment is $5.45 a week, ride-on toys are $8.00 a week and sport toys are $6.50 a week.*
>
> *Dress-ups have a size (a string), and a genre (e.g. "super-hero") and a count of how many pieces make the costume (for example a costume consisting of pants, top and mask would be 3 piece). The entire costume has a laundry fee of $3.00. The cost of hiring a costume is the ((piece count * $3.50) * number of weeks) + laundry fee.*
>
> *For Play equipment we've tried various ways to calculate costs but in the end decided they are so varied that we couldn't develop a formula. Instead we have found by trial and error for each play equipment a weekly cost that gives us a good hiring frequency and return on investment. This cost should be stored when adding a piece of play equipment into the system. We also record the weight*

in kilograms of the equipment and the dimensions (height, width, depth) in centimetres as that may be important for some customers.

The display of a specified item should produce neatly formatted output of data relevant for that type of item."

Important: Your supervisor advises that *Item* should be extended to implement classes *Toy*, *DressUp* and *PlayEquipment* to implement the requirements described in the above interviews.

Your Stage *B* program must utilise the same array (*holdings*) used in Stage *A* for storing information about all the items, in the class *StageB*. Your supervisor also specifies that the *Item* class should be modified so only subclasses of *Item* can be instantiated and that those subclasses cannot be further extended. In addition, the *Item* class must be modified so as to force all subclasses to implement a method *determinePrice()* which returns a double (the cost) relevant for the subclass, calculated as per Kanchan's interview above. You should assume when adding a new toy to the system, the manager will enter, as a string, a valid toy category (one of *"construction", "ride-on" or "sport"*) and this information should be stored by the *Toy* object and used by it to determine the hire price. You are expected to adhere to all relevant object-oriented programming guidelines as described in stage *A.*

Stage *C* – Adding Exception Handling

In this stage you rename class *StageB* as *StageC*, modify and extend the *Item* class, modify the *Toy* class and modify the *StageC* class to meet the requirements presented in the supervisor requirements below.

Your supervisor specifies that the *Toy* constructor must ensure the toy category is one of those specified in Kanchan's interview as you can no longer assume the user will enter a valid toy category. The *Toy* constructor must reject an invalid toy category parameter, and throw an *IllegalArgumentException* with a suitable message. This exception must be caught in *StageC* and the message displayed.

Your supervisor has also asked you extend class *Exception* in order to define your own exception type, to be called *HiringException.* This *HiringException* class should be in its own separate source code (*.java*) file. The *Item hireItem(customerID,dueDate)* and *returnItem()* methods must be rewritten to be 'void' and to generate and throw a *HiringException* containing a suitable error message if an attempt to hire or return, respectivey, the item fails. Note that any *HiringException* that is thrown should be caught in *StageC* and the message displayed.

Stage *D* – Adding Object Persistence and re-factoring holdings to an ArrayList

In this stage you rename class *StageC* as *StageD*, modify and modify the *Item, DressUp, Toy, PlayEquipment* and *StageD* classes to meet the requirements presented in the interview and supervisor requirements below.

Kanchan – ChildzPlay Manager

"I'd like the system to save all information before exiting, and then restore that information again when it is next run."

Your supervisor instructs you to refactor array *holdings* to a single ArrayList of type *Item* called *holdings*. There is no longer a limit on the number of items that can be stored.

Your supervisor also instructs you to design an approach to writing out to file information for all items currently stored in the system in such a way that they can be reconstructed in full and then implement functionality to:

- extract details for all items currently in the system and write the information out to a text file (in a format which will allow the same set of objects to be restored in full later) when the user chooses to exit the program, and
- read the information back in from the text file and use it to restore the existing set of objects in full and store them back in the *holdings* ArrayList when the program starts up.

The design should take advantage of inheritance. Your text file location should be relative to your project workspace – it should not be necessary to specify a full path and your solution will lose marks if it opens a file with a path that exists on your local machine, rather than one relative to the project workspace. If there is no data file then the program should proceed to run in an "empty" state, asking the user what the maximum number of items to store is, and then displaying a menu for providing the stage A/B/C functionality.
You are **not** permitted to use automatic object serialization in your implementation.

Referencing guidelines

What: This is an individual assignment, and all submitted code must be your own. If you have used examples of code from sources other than the contents directly under the course Canvas Modules link, or the recommended textbook (e.g. an example from the web or other resource) as the basis for your own code then you should acknowledge the source(s) and give references using the IEEE referencing style.

Where: Add a block code comment near the work to be referenced and include the reference in the IEEE style.

How: To generate a valid IEEE style reference, please use the citethisforme tool if unfamiliar with this style.

Submission format

Submit your Java course code file (or Eclipse project) on the course Canvas under the Assignments/Assignment3 link as a file upload.

Marks are awarded for meeting requirements as closely as possible. Clarifications/updates may be made via announcements/relevant discussion forums on the course Canvas.

Academic integrity and plagiarism

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas.

You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods

- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites

If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct.

Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the University website.

Assessment declaration

When you submit work electronically, you agree to the assessment declaration.

RMIT UNIVERSITY

Criteria	Categories						Pts
Code style/documentation	All three *Style Guide* guidelines were implemented, no areas of improvement were identified	All three *Style Guide* guidelines were implemented, minor issues noted with one guideline	All three *Style Guide* guidelines were implemented, minor issues noted with two guidelines	Significant issues noted with one guideline	Major issues were noted with two guidelines	None of the *Style Guide* guidelines were not followed	
	2.0 pts	**1.5 pts**	**1.0 pts**	**0.5 pts**	**0.1 pts**	**0.0 pts**	**2.0**
Item class modelling	*Item* class was implemented with attributes, methods and constructor that met stage D requirements of this class.	*Item* class was implemented with attributes, methods and constructor that met stage C requirements of this class.	*Item* class was implemented with attributes, methods and constructor that met stage B requirements of this class.	*Item* class was implemented with attributes, methods and constructor that met all stage A requirements of this class.	*Item* was attempted but attributes or methods or constructor were inadequate to implement stage A requirements.	*Item* class wasn't implemented	
	3.0 pts	**2.5 pts**	**2.0 pts**	**1.0 pts**	**0.5 pts**	**0.0 pts**	**3.0**
StageA/B/C/D class functionality	*StageD* met all requirements for stage D.	*StageC/D* met all requirements for stage C.	*StageB/C/D* met all requirements for stage B.	*StageA/B/C/D* met all requirements for stage A.	*StageA/B/C/D* met some, but not all requirements for stage A.	Functionality not implemented	
	6.0 pts	**4.5 pts**	**3.5 pts**	**2.0 pts**	**1.0 pts**	**0.0 pts**	**6.0**
Item subclasses	*Item* subclasses *DressUp, Toy* and *PlayEquipment*	*Item* subclasses *DressUp, Toy* and *PlayEquipment* implemented to	*Item* subclasses *DressUp, Toy* and *PlayEquipment* implemented to	At least two of the *Item* subclasses *DressUp, Toy* or *PlayEquipment*	One of the *Item* subclasses *DressUp, Toy* or *PlayEquipment*	Functionality not implemented	

	implemented to meet stage D requirements.	meet stage C requirements.	meet stage B requirements.	attempted, with either missing subclasses or significant class errors or omissions observed.	attempted, with either missing subclasses or significant class errors or omissions observed.		
	5.0 pts	**4.0 pts**	**3.0 pts**	**2.0 pts**	**1.0 pts**	**0.0 pts**	**5.0**
HiringException and IllegalArgumentException Handling	*HiringException* defined as per Stage *C* requirements. *Item* methods *hireItem(customerID, dueDate)* and *returnItem()* re-factored as per Stage C requirements, *StageC/D* class catches and correctly processes *HiringException* as per stage C requirements. Constructor of *Toy* subclass correctly generates and throws *IllegalArgumentException* and StageC/D correctly catches and processes it as per stage C requirements.	*HiringException* defined as per Stage *C* requirements. Minor issue noted with one of: *Toy* constructor correctly generating and throwing *IllegalArgumentException* or with refactoring of *Item hireItem* and *returnItem* methods to throw *HiringExceptions* or with correctly catching and processing exceptions in Stage C/D.	*HiringException* defined as per Stage *C* requirements. Minor issue noted with one of: *Toy* constructor correctly generating and throwing *IllegalArgumentException* or with refactoring of *Item hireItem* and *returnItem* methods to throw *HiringExceptions* or with correctly catching and processing exceptions in Stage C/D.	*HiringException* defined as per Stage *C* requirements. Significant issue noted with one of: *Toy* constructor correctly generating and throwing *IllegalArgumentException* or with refactoring of *Item hireItem* and *returnItem* methods to throw *HiringExceptions* or with correctly catching and processing exceptions in Stage C/D.	Some evidence of an attempt to define *HiringException* type but code is not functional.	Functionality not implemented	
	4.0 pts	**3.5 pts**	**3.0 pts**	**2.0 pts**	**1.0 pts**	**0.0 pts**	**4.0**
Object Persistence and refactoring *holdings* to an ArrayList	Stage D object persistence functionality and refactoring of *holdings* to an ArrayList correctly	Stage D object persistence functionality correctly implemented but one of the following issues	Stage D object persistence functionality correctly implemented but two of the following issues	Stage D object persistence functionality correctly implemented but all of the following issues noted: not	Some attempt to add object persistence. All of the following issues noted: not taking advantage of inheritance in	Functionality not implemented	

	implemented, no issues noted.	noted: not taking advantage of inheritance in the design for object persistence, not initially running in an 'empty' state if the data file doesn't exist, not specifying a file location relative to the project workspace, not refactoring *holdings* array to an ArrayList.	noted: not taking advantage of inheritance in the design for object persistence, not initially running in an 'empty' state if the data file doesn't exist, not specifying a file location relative to the project workspace, not refactoring *holdings* array to an ArrayList.	taking advantage of inheritance in the design for object persistence, not initially running in an 'empty' state if the data file doesn't exist, not specifying a file location relative to the project workspace, not refactoring *holdings* array to an ArrayList.	the design for object persistence, not initially running in an 'empty' state if the data file doesn't exist, not specifying a file location relative to the project workspace, not correctly writing/reading file to implement stage D functionality.		
	3.0 pts	**2.5 pts**	**2.0 pts**	**1.5 pts**	**0.5 pts**	**0.0 pts**	**3.0**
Object-oriented technique	All object-oriented guidelines specified in stage *A* were adhered to.	Object-oriented guidelines, specified in stage *A*, were generally adhered to with a single minor issue noted.	Object-oriented guidelines, specified in stage *A*, were generally adhered to with two minor issues noted.	Object-oriented guidelines, specified in stage *A*, were generally adhered to with one significant issue noted.	Object-oriented guidelines, specified in stage *A*, were attempted, with multiple significant issues noted.	Object-oriented techniques were not implemented.	
	2.0 pts	**1.5 pts**	**1.0 pts**	**0.5 pts**	**0.1 pts**	**0.0 pts**	**2.0**
						Total:	**25 pts**