COSC1284 Programming Techniques
Semester 2 (2019)
Assignment 2

Ref: http://pngimg.com/uploads/shopping_cart/shopping_cart_PNG66.png

Due: **Friday 11 October, 2019**
09:00 p.m. AEST
i.e. End of week 11.

Late submissions accepted until
Wednesday 16 October, 2019
09:00 p.m. AEST

Assignment Type: **Individual (Group work is not permitted)**

Total Marks: **15 marks (15%)**

Background information

For this assignment you need to write an object-oriented console application in the Java programming language which adheres to basic object-oriented programming principles shown below:

a) Your code should follow good object-oriented principles such as: encapsulation, composition, cohesion.

b) Setting the visibility of all instance variables to private.

c) Using getter and setter methods only where needed and appropriate with consideration given to scope (visibility).

d) Only use static attributes and methods when you have good reason to do so, such as reducing code repetition whilst still maintaining good objected oriented design.

e) Using superclass methods to retrieve and/or manipulate superclass properties from within subclass methods. Avoiding direct access to class attributes.

f) Taking advantage of polymorphism wherever possible when invoking methods upon objects that have been created and avoiding unnecessary casting.

Business Rules:

Each class must implement the following business rules in addition to the class specific business rules detailed in the specification.

It is not possible to assign an empty string to an attribute, instead "N/A" should be assigned.

Any numeric values that are considered invalid should be assigned a value of -1.

You must ensure that appropriate protection from null values is considered by: preventing attributes from having null values and/or check for the presence of null values prior to performing an operation on an object.

Arrays should be appropriately size and re-sized to ensure they are only large enough to accommodate the number of items the array holds.

All classes that represent a single object must implement both a 'getDetails' and a 'toString' method to provide string representations of the state of an object.

Overview

The staff at **MiBay** want you to build a program that will allow them to manage the creation and delivery of packages of goods.

A user will be able to:

- Search for packages by delivery date.
- Display deliveries sorted in order of the customer surname.

You will be addressing these requirements by implementing a series of classes designed to meet these needs. The classes must be designed according to object-oriented principles.

Assignment 2 – Getting Started

This assignment is divided up into several stages, each of which corresponds to the concepts discussed during the course as shown below:

• Stage 1 - Package & associated classes	(Inheritance)	(5 marks)
• Stage 2 MiBay System	(Algorithms, Branching & Repetition)	(5 marks)
• Stage 3 - Exception Handling	(Exception Handling)	(2 marks)
• Stage 4 - Persistence to file	(File Handling)	(3 marks)

The assignment is designed to increase in complexity with earlier stages being easy and the later stages being more difficult. In addition, the specification will be more prescriptive in the earlier stages. The later stages of the assignment are less prescriptive and will require you to be more independent and resolve design issues on your own.

You are not permitted to use streams from Java 1.8.
You must use an array to store any collections of data in your program.
You must not use ArrayLists, Maps or any other data structures within the Java API or from 3rd party libraries.
Failing to comply with these restrictions will incur a 50% marking penalty.

Disclaimer:
While the scenario described is based upon the concept of a package delivery service, the specification for this task is intended to represent a simplified version of such a system and it is not meant to be a verbatim simulation of any such package delivery service.

Stage 1 of the Assignment begins on the next page ...

Stage 1 - Package Hierarchy (and associated classes) (5 marks)

This stage requires implementing several classes. Some of the classes have been provided below; others you will be expected to create as needed.

As a minimum you will need to create two classes called **Package & PlatinumPackage**

You will be required to think carefully about the concepts you have learned in the course and make decisions in regard to modifying the classes to meet the stated requirements.

The **Package** and **PlatinumPackage** classes share common attributes and behaviours as they both represent a package. However, there will be additional attributes and behaviours that belong only to a **PlatinumPackage's**.

A **PlatinumPackage is a 'Package'**. + additional attributes & methods.

In object-oriented programming whenever we see an **'is a'** relationship, we model it by implementing a class hierarchy.

A) Create a Customer and associated classes

Your class should have the following method signature as a minimum.

```
public Customer(String firstName, String lastName, Address address)
```

Customer specific business rules:

1. The customer must have a first and a last name.
2. The customer must have at least one address.
3. There may be up to 2 addresses. If a second address is provided it will be considered an alternate delivery address.

You should write a test class that will **FULLY** test this class and its dependencies.

Think about the following scenarios as a starting point.

1. Valid construction

2. Invalid construction

- Empty Strings
- Null values

3. Operations

- Setting a preferred alternate address when it does not exist
- Adding a second alternate address
- Setting a preferred alternate address when it does exist.
- Adding a third address

B) Create a Product class

You class should have the following method signatures as a minimum.

```
public Product(String name, double weight, double cost)
```

Business Rules:

1. The weight of the product must be greater than zero.
2. The cost of the product must be greater than one.

You should write a test class that will **FULLY** test this class.

Think about the following scenarios as a start point.

1. Valid construction

2. Invalid construction

- Empty Strings
- Null values
- Zero values

C) Create Package class

You class should have the following method signatures as a minimum.

```
public Package(Customer customer, Product product)

public boolean addProduct(Product product)

public boolean removeProduct(Product product)
```

Business Rules:

1. There must be only one customer.
2. There must be at least one product.
3. Products can be added to the list of products after construction.
4. Products are removed from the list of products based on the name of the product supplied.
5. If more than one product has an exact match to the name supplied, only one instance of the product should be removed from the array.

You should write a test class that will **FULLY** test this class.

Think about the following scenarios as a start point.

1. Valid construction

2. Invalid construction

3. Adding/Removing products

D) Create PlatinumPackage class (sub-class of Package)

You class should have the following method signatures as a minimum.

```
public PlatinumPackage(Customer customer, Product product,
                       String memberNumber)

public boolean updateMemberNumber(String memberNumber)
```

Business Rules:

1. All the business rules that apply to the parent 'Package' class.
2. Must have a member number.
3. The member number must be 10 alphanumeric characters. The initial characters should be alphabetic, and the subsequent characters should alternate between alphabetical and numerical.

   For example:

   Valid Member Numbers:

   ```
   "D0B4M4S7Y5"
   "Z0W0P9J2V6"
   "F9N5S7H9Z1"
   "P5N3M1R9Z8"
   "Z0L3C8J9L8"
   "Q5G7J8F8K3"
   "H8Z9L9A4E9"
   "U8M4Z3N9F5"
   "I9X4L5A5M0"
   "X9S9V6I0Y8"
   ```

   Invalid Member Numbers:

   ```
   "D0B4M4S7Y"      (9 characters)
   "D0B4M4S7Y5A"    (11 characters)
   "D0BM44S7Y5"     (non-alternating)
   "D$B4M4S7Y5"     (non-alphanumeric)
   ```

You should write a test class that will **FULLY** test this class.

Think about the following scenarios as a start point.

1. Valid construction

2. Invalid construction

3. Updating a member number

Do not continue with the next stage until you have fully tested your program to ensure it is operating correctly.

Stage 2 - Menu & MiBayApplication class (basic functionality) (5 marks)

Make a backup of your program before continuing

YOU MUST COMPLETE STAGE 1 BEFORE ATTEMPTING STAGE 2

IF STAGE 1 HAS NOT BEEN COMPLETED, STAGE 2 WILL NOT BE ASSESSED

Here, you will begin with the implementation of the **MiBayApplication** application class, which will use an array of **Delivery** references called **'deliveries'** to store and manage all the deliveries that are added to the system.

1.) You should:

A) create a class called **'MiBayApplication'** for managing all the deliveries.

B) create a class to present a menu to the user for interacting with the program. **(printing should only be done in the menu class. No other classes in the system are permitted to print to the console)**

2.) The menu options that must be presented to the user are:

```
*** MiBayApplication System Menu ***

Add Customer                                    AC
Add Product                                     AP
Prepare Order                                   PP
Display ALL Deliveries (Sorted by Name)         DA
Delivery Search (display deliveries on date)    DS
Seed Data                                       SD
Exit Program                                    EX
Enter selection:
```

You must implement the menu as shown. **Do not change the options** or the inputs for selecting the menu options.
(Note: the letters on the right represent what the user must type to use that feature.
In other words, to create a new Delivery, the user must input 'CD'. The solution should be case insensitive, that is the user should be able to enter either ‘cd’ or ‘CD’ or ‘cD’, etc.)

Your task is to work on this initial **MiBayApplication** class and the **Menu** class by implementing the features shown in the menu above. A description of the functionality that needs to be implemented for each of these features is provided below.

A) Add customer

This first feature **'Add Customer'** should prompt the user to enter all relevant details for a **Customer**. The customer should be created and stored in the system. This will allow for customers to be selected later when creating an order.

Example Outputs

```
Example:
Enter customer details:
Enter first name:       Henry
Enter last name:        Cavill
Enter street number:    83
Enter street name:      Dalgliesh Street
Enter suburb:           South Yarra
Enter postcode:         3141

Henry Cavill was successfully added to the system.
```

B) Add product

This second feature '**Add Product**' should prompt the user to enter all relevant details for a **Product**. The product should be created and stored in the system. This will allow for products to be selected for later when creating an order.

Example Outputs

```
Example:
Enter product details:
Enter name:             Man of Steel DVD
Enter weight:           300
Enter cost:             24.99

Man of Steel DVD was successfully added to the system.
```

C) Prepare Order

This third feature **'Prepare Order'** should prompt the user to enter all relevant details for an order.

```
Example:
Please choose a customer from the list:
1. Henry Cavill
2. Matthew Broderick
Input: 1

Please choose a product from the list:
1. Man of Steel
2. The Lion King
Input: 1

Please enter the delivery date:
Enter Day: 15
Enter Month: 7
Enter Year: 2019

Would you like to add another product? (Y/N)
Input: Y

Please choose a product from the list:
1. Man of Steel
2. The Lion King
Input: 2

The Lion King was successfully added to the order.
```

```
Would you like to add another product? (Y/N)
Input: N

Is this a Platinum Package (Y/N)?:
Input: N

Package for Henry Cavill was successfully prepared.
```

Example:

```
Please choose a customer from the list:
1. Henry Cavill
2. Matthew Broderick
Input: 1

Please choose a product from the list:
1. Man of Steel
2. The Lion King
Input: 1

Please enter the delivery date:
Enter Day: 15
Enter Month: 7
Enter Year: 2019

Would you like to add another product? (Y/N)
Input: Y

Please choose a product from the list:
1. Man of Steel
2. The Lion King
Input: 2

The Lion King was successfully added to the order.


Would you like to add another product? (Y/N)
Input: N

Is this a Platinum Package (Y/N)?:
Input: N

Please input your member number: D0B4M4S7Y5

Package for Henry Cavill was successfully prepared.
```

The user should not be prompted unnecessarily. For example if the customer list is empty then the menu should not prompt for the list of products.

Example:

```
Enter your selection:
Input: PO

Sorry no customers available.
```

Example:

```
Enter your selection:
Input: PO

Please choose a customer from the list:
1. Henry Cavill

Input: 1
Sorry no products available.
```

C) Display ALL Deliveries (Sorted by customer last name)

This feature should begin by prompting the user for the order in which they want the details displayed.

The choice is between ascending and descending order. The sort order is based on the last name of the customer. The full details of each object should be printed to the console.

NOTE: You must implement your own sorting logic. You are not permitted to use any built-in functions or libraries to achieve the sort. Failure to implement your own sorting will result in a 20% marking penalty.

Example:

```
Enter sort order (A/D): D
Summary of all packages:

_____________________________________
Name:           Henry Cavill
Address:        83 Dalgliesh Street
                South Yarra 3141
Products Ordered:
Name:           Man of Steel
Weight:         300g
Cost:           $24.99

Name:           The Lion King
Weight:         320g
Cost:           $17.99
_____________________________________
Name:           Matthew Broderick
Address:        42 Pride Avenue
                Elwood 3184
Products Ordered:
Name:           Man of Steel
Weight:         300g
Cost:           $24.99
```

Example:

```
Enter sort order (A/D): D
Summary of all packages:

_____________________________________
Name:           Henry Cavill (D0B4M4S7Y5)
Address:        83 Dalgliesh Street
                South Yarra 3141
Products Ordered:
Name:           Man of Steel
Weight:         300g
Cost:           $24.99

Name:           The Lion King
Weight:         320g
Cost:           $17.99
_____________________________________
Name:           Matthew Broderick
Address:        42 Pride Avenue
                Elwood 3184
Products Ordered:
Name:           Man of Steel
Weight:         300g
Cost:           $24.99
```

D) Delivery Search (display deliveries on date)

This feature should begin by prompting the user for a date. The full details of each delivery that is scheduled for the date provided should be printed to the console.

NOTE: You must implement your own filtering logic. You are not permitted to use any built-in functions or libraries to achieve the filtering. Failure to implement your own filtering will result in a 20% marking penalty.

Example:

```
Enter your selection:
(Hit enter to cancel any operation)
so

Enter Day:              13
Enter Month:            7
Enter Year:             2019

_____________________________________________
Name:           Henry Cavill
Address:        83 Dalgliesh Street
                South Yarra 3141
Delivery Date:  13/07/2019
Products Ordered:
Name:           Magic Mike
Weight:         336g
Cost:           $26.99

_____________________________________________
Name:           Whoopi Goldberg
Address:        57 Elaine Court
                St Albans 3021
Delivery Date:  13/07/2019
Products Ordered:
Name:           Mr and Mrs Smith
Weight:         355g
Cost:           $25.99
```

Example:

```
Enter your selection:
(Hit enter to cancel any operation)
so

Enter Day:              13
Enter Month:            7
Enter Year:             2019

_____________________________________________
Name:           Henry Cavill (D0B4M4S7Y5)
Address:        83 Dalgliesh Street
                South Yarra 3141
Delivery Date:  13/07/2019
Products Ordered:
Name:           Magic Mike
Weight:         336g
Cost:           $26.99

_____________________________________________
Name:           Whoopi Goldberg
Address:        57 Elaine Court
                St Albans 3021
Delivery Date:  13/07/2019
Products Ordered:
Name:           Mr and Mrs Smith
Weight:         355g
Cost:           $25.99
```

E) Menu System & Seed Data method

You should implement a method called '**seedData'** that pre-populates the system with a range of customers and products.

When the seed data menu item is selected:

4 hard coded customers and 4 hard coded products should be populated into the arrays.

If this option is not selected at runtime, then the collection of meals and deliveries should be empty.

If the collection of meals and deliveries already have data, then this method SHOULD ADD the seeded data to the current data.

If the collection has already been seeded, then the seed data should not be added again.

You must ensure that your program does not crash under any circumstances, so you will need to thoroughly test your program before final submission.

Stage 3 - Exception handling (2 marks)

Make a backup of your program before continuing

YOU MUST COMPLETE STAGE 2 BEFORE ATTEMPTING STAGE 3

IF STAGE 2 HAS NOT BEEN COMPLETED, STAGE 3 WILL NOT BE ASSESSED

It has been identified that there are potential issues with the program that cannot be prevented but must be dealt with at the time the program is running.

You should comply with these requests by making the following changes to your program.

NOTE: When you implement exceptions into your program you will need to modify the other classes in your program to work with exceptions.

Exceptions will be **generated and thrown** in the class that defines the business rule that caused the error.

Exceptions will be caught and handled in the **Menu** and/or **MiBayApplication** system classes.

A) Creating the Exception Classes

Define your own custom Exception subclasses for each of the business object classes such as Address, Customer, Product, and Package to handle any violations of the business rules as defined in these classes.

B) Generating Exceptions

Whenever a business rule is violated, you will now generate an exception instead of setting a default value as indicated in earlier in the specification.

C) Handling Exceptions

These various custom exceptions should then be allowed to propagate back up to the **MiBayApplication & Menu** system classes (i.e., the exception **should not be caught locally** where the exception is generated).

Any exceptions will need to be caught and handled in an appropriate manner in the **MiBayApplication or Menu** system class (by displaying the error message contained within the various exception objects that have been propagated up from the relevant method call).

You can simply display the exception message and return to the menu. You are not required to make users re-enter invalid input.

You should also handle any invalid numeric input in the menu class that generates a system exception.

For example:

If the menu prompts the user to enter a number, such as a cost, your program should be able to handle the situation where the user incorrectly inputs 'one' instead of 1.

Stage 4 - Persistence (File I/O) (3 marks)

Make a backup of your program before continuing

YOU MUST COMPLETE STAGE 3 BEFORE ATTEMPTING STAGE 4

IF STAGE 3 HAS NOT BEEN COMPLETED, STAGE 4 WILL NOT BE ASSESSED

This section of the program is designed to challenge and extend your ability to take a problem and work out how to solve it.

It is expected that **you will design your own solution** for this feature.

Your program should incorporate file handling functionality so that it writes the details for each **Customer (including address) & Product** object currently in the **MiBayApplication** system out to file when the program terminates.
(i.e., when the user selects the "Exit" option in the menu).

You only need to write out the customer (including address) and product objects in the program, you are not required to write out the full details of all the packages.

The data that was previously written out to file should then be read back in automatically when the program is started up again and restore the program to the state it was in when the program exited.

This feature should write the data out to two files. One is the main data file, the other will be a duplicate copy to act as a backup of the data.

If the main data file is not found in the local folder then the program should:

1.) First check for the presence of a backup file and if found use this backup file for loading the data.

2.) If loading from the backup file, display a message indicating that the data was loaded from a backup file.

3.) If no backup file is found, display a message indicating that no data was loaded and continue to the program menu without reconstructing any objects.

The format that you write the data out in is entirely at your own discretion so long as it is written to a text file.

Hint: You could use the toString method :-)

You must NOT use object serialisers to accomplish this task. Using object serialisation in this section will result in a zero grade for this section.

You can make any changes or include any additional methods that you deem necessary to facilitate the writing out and reading in of the program state.

Note that the program design described earlier in the specification may not fully support what is required in this stage, so part of this task is identifying what aspects need to be considered when designing / implementing a solution.

Remember that you will still be required to adhere to basic object-oriented programming principles (e.g., encapsulation, information hiding, etc.) when designing your solution for this aspect of the program.

IMPORTANT NOTES:

These file reading / writing features are advanced functionality.

Marks will only be awarded for this section if the following conditions are met:

A) You must have completed all the previous stages before attempting this section) Non-functional code or code which needs to be commented out in order to get the rest of the program to compile / run will receive zero marks in this section, as will code that just writes data out without any real intent of structuring that data to facilitate object persistence.

C) Also note that **you are not permitted to use automatic serialisation or binary files** when implementing your file handling mechanism - you must use a **PrintWriter** for writing data out to a text file and a **BufferedReader** or **Scanner** when reading the data back in from the same text file that data was written out to previously.

Coding Style

Your program should demonstrate appropriate coding style, which includes:

- **Formatting**

  Indentation levels of 3 or 4 spaces used to indent or a single tab provided the tab space is not too large - you can set up your IDE/editor to automatically replace tabs with levels of 3 or 4 spaces.

  A new level of indentation added for each new class/method/ control structure used.

  Indentation should return to the previous level of at the end of a class/method/control structure (before the closing brace if one is being used). Going back to the previous level of indentation at the end of a class/method/control structure (before the closing brace if one is being used)

  Block braces should be aligned and positioned consistently in relation to the method/control structure they are opening/closing.

  Lines of code not exceeding 80 characters in length - lines which will exceed this limit are split into two or more segments where required (this is a guideline - it's ok to stray beyond this by a small amount occasionally, but try to avoid doing so by more than 5-6 characters or doing so on a consistent basis).

  Expressions are well spaced out and source is spaced out into logically related segments

- **Good Coding Conventions**

  Identifiers themselves should be meaningful without being overly explicit (long) – you should avoid using abbreviations in identifiers as much as possible (an exception to this rule is a “generic” loop counter used in a for-loop).

  Use of appropriate identifiers wherever possible to improve code readability.

  All identifiers should adhere to the naming conventions discussed in the course notes such as ‘camel case’ for variables, and all upper case for constants etc.

  Complex expressions are assigned to meaningful variable names prior to being used to enhance code readability and debugging.

Commenting

Note: the examples provided below do not reflect actual comments you should have in your assignment but are used for demonstration purposes only.

Class Level commenting

Each file in your program should have a comment at the top listing your name, student number and a brief description of the contents of the file (generally stating the purpose of the class)

```
/**
* <h1>Menu</h1>
* <p><b>This class provides a user interface to allow the user * to
use different examples of algorithms for sorting and
* filtering data.</b></p>
* <p><b>References: https://www.baeldung.com/java</b></p>
*
* @author  Student Name (s1234567)
* @version 1.0
* @since   2019-05-16
*/
```

Method Level commenting

All methods should be commented using Java docs.

```
/**
  * <p>This method enables the insertion of indentation
  * levels into a string.</p>
  * <p>The method builds a string with tab characters equal to
  * the value provided.</p>
  * @param level This provides number of indents required.
  * @return String This returns a string with the specified
  * number of tab characters.
  */
```

Complex methods should have an algorithm as a multi-line comment. You don't need to specify test cases for every algorithm, only if it is appropriate.

```
/**
  * <b>This demonstrates how to calculate the sum of all
  * numbers from zero up to a given value.</b>
  * <p>This method uses iteration to perform the calculation.
  * Negative numbers are not supported.</p>
  * <pre>
  *    ALGORITHM - Iterative sum
  *    BEGIN
  *       DEFINE the upper bound
  *       DEFINE container for storing the result
  *       IF the upper bound less than zero
```

```
 *          EXIT
 *       END IF
 *       REPEAT upper bound number of times
 *          ADD upper bound to the current result
 *       END REPEAT
 *    END
 * </pre>
 * @param upperBound This provides the upper bound of the
 * numbers being processed.
 * @return int This sum of all the numbers between zero and
 * the upper bound.
 */
```

Code block commenting

Methods that contain multiple logic or code blocks should either be refactored into smaller discrete methods or provide inline commenting to identify and explain each logic or code block.

```
/* calculate the total value of the room rental
 * prior to applying surcharge.*/
int standardRate = 200;
int numberOfNights = 4;
int totalBeforeSurcharge = standardRate * numberOfNights;
```

Commented out code

Any incomplete or non-functional code should be removed from your implementation prior to your final submission.

Code that is not being used in your final implementation will attract marking penalties.

Examples of bad commenting

An example of a bad comment and/or coding style:

```
// declare an int value for storing the basic nightly rate for a
room
int standardRoomRate = 200;
// declare and assign an initial account balance.
double x = 500.0;
```

What to Submit

You are being assessed on your ability to write a program in an object-oriented manner in this assignment. Writing the program in a procedural style is not permitted and will be considered inadmissible receiving a zero grade.

You should stick to using the standard Java API (version 1.8) when implementing your program. Note the restrictions noted earlier in this document in relation to the Java Collection Framework, Streams and the use of 3rd party libraries.

You should export your entire eclipse project to a zip archive and submit the resulting zip file - do this from within eclipse while it is running, not by trying to copy or move files around in the eclipse workspace directly, as you may corrupt your entire workspace if you do something wrong.

All students are also advised to check the contents of their zip files by opening them and viewing the files contained within before submitting to make sure they have done it correctly and that the correct (latest) version of the source code file is present to avoid any unpleasant surprises later on.

You should also download a copy of your submission from Canvas after submission, so that you can double check that you have submitted the correct version.

Technical issues that result in the loss of your work or submitting an incorrect version **WILL NOT** be considered a basis for extensions or re-marking.

We are obliged to accept each submission in the form it is sent to us in, so make sure you submit the correct final version of your program!

The name of both your Eclipse project and your zip archive should be your student number followed by an underscore, then append A2. For example:

Project Name: s123456_A2
Zip Archive: s123456_A2.zip

Please note if you make multiple submissions, canvas will change the name of the file you submit.

That is OK and you should not worry about the name change.