**INTI International College Penang School of Computing**

**3+0 Bachelor of Science (Hons) in Computer Science, in collaboration with Coventry University, UK 3+0 Bachelor of Science (Hons) in Computing, in collaboration with Coventry University, UK**

# Coursework cover sheet

**Section A - To be completed by the student.**

|  |  |
| --- | --- |
| Full Name: JAYASHREE A/P ELUMALAI | |
| CU Student ID Number: 14196184 | |
| Semester:1 | |
| Session:  **April 2023** | |
| Lecturer:  **Puteri Nursyawati Azzuri (puteri.azzuri@newinti.edu.my)** | |
| Module Code and Title:  **4067CEM Software Design** | |
| Assignment No. / Title:  **Continuous Assessment** | % of Module Mark:  **50** |
| Hand out Date:  **12 May 2023** | Due Date:  **Task 3: 7 July 2023, by 11.59pm.** |
| Penalties: No late work will be accepted. If you are unable to submit coursework on time due  to extenuating circumstances, you may be eligible for an extension. Please consult the lecturer. | |
| Declaration: I/we the undersigned confirm that I/we have read and agree to abide by the University regulations on plagiarism and cheating and Faculty coursework policies and procedures. I/we confirm that this piece of work is my/our own. I/we consent to the appropriate storage of our work for plagiarism checking.    Signature(s): | |

# Section B - To be completed by the module leader

|  |  |  |
| --- | --- | --- |
| Intended learning outcomes assessed by this work:   1. Understand and apply appropriate concepts, tools, and techniques to each stage of the software development. 2. Understand and apply design patterns to software components in developing new software. 3. Demonstrate an understanding of project planning and working to agreed deadlines, along with professional, interpersonal skills and effective communication required for software production.   5. Demonstrate an awareness of, and ability to apply, social, professional, legal, and ethical standards as documented in relevant laws and professional codes of conduct such as that of  the Malaysian National Computer Confederation. | | |
| Marking scheme | Max | Mark |
| 1. User Story Mapping | 20 |  |
| 2. Setting up a GitHub |  |
| Repository | 10 |
| 3. Creating a Class diagram and |  |
| design pattern selection | 30 |
| 4. Creating a Prototype User |  |
| Interface and Usability Testing | 20 |
| 5. Discuss the ethical issue |  |
| related to the software | 20 |
| Total | 100 |  |

Class Diagram

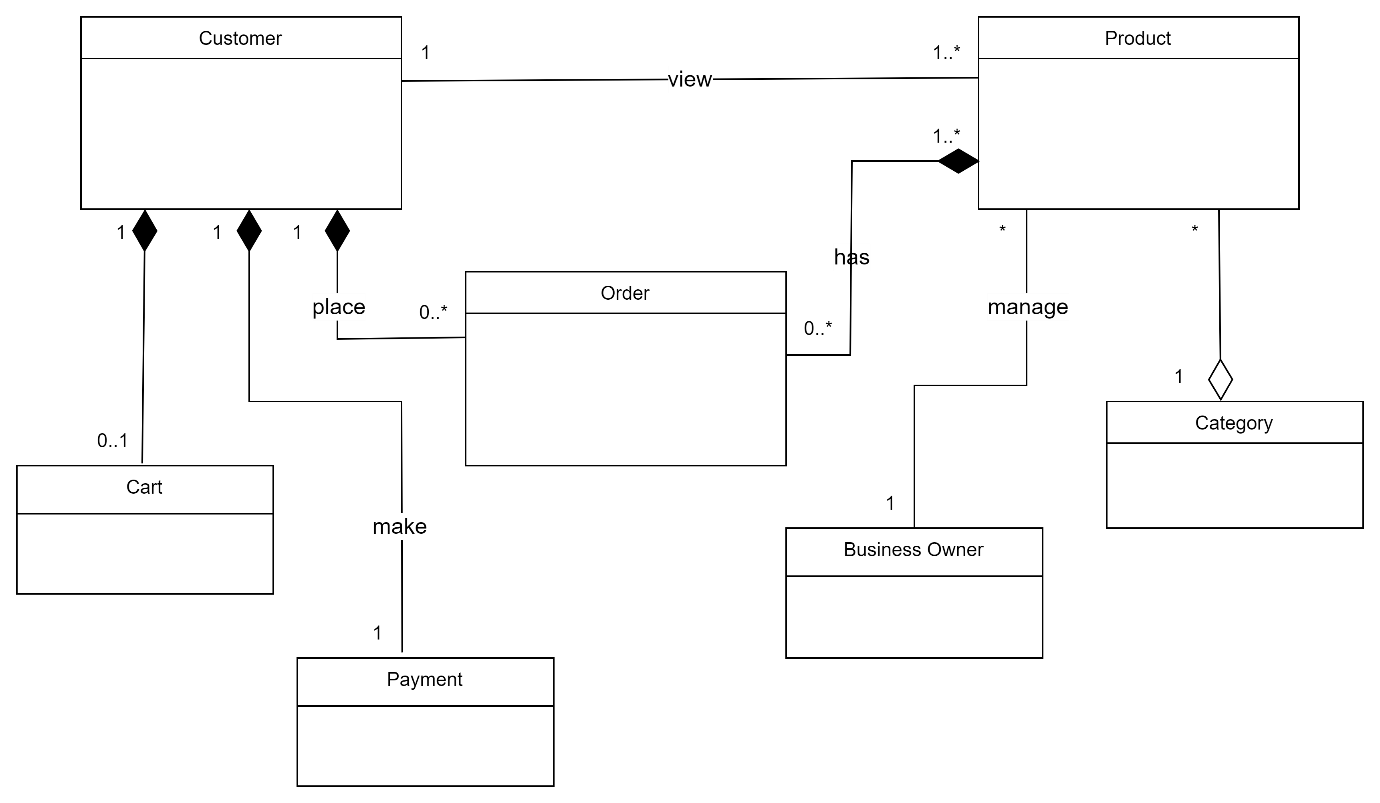


Figure 1

Figure 1 shows the class diagram created for the clothing business system. There are seven classes which are: Customer, Cart, Payment, Order, Product, Category and Business Owner. The ‘Customer’ represents the users of the system. The ‘Cart’ represents a shopping cart that a customer can use to add products before placing an order. The ‘Payment’ represents the payment made for the order. The ‘Order’ represents the order for products placed by the customer. The ‘Product’ represents a clothing product available for ordering. The ‘Category’ represents the category of clothing, such as men's clothing, women’s clothing and others. The ‘Business Owner’ represents the owner of the clothing business.

The Customer class can view the products in the Product class and they have an association relationship which is depicted by the solid line. Each ‘Customer’ is associated with 1 or more ‘Product’ and each ‘Product’ is associated with exactly one ‘Customer’. The Customer class can make payments with the Payment class. They have a composition relationship which is depicted by a solid line and a filled diamond-shaped arrowhead pointing from the composite class (Customer) to the component class (Payment). The composite class (Customer) represents the whole concept and the component class (Payment) represents part of the whole. They have a composition relationship because if the Customer class is destroyed then the Payment class will also be destroyed. Each ‘Customer’ is composited with exactly one ‘Payment’ and each ‘Payment’ is composited with exactly one ‘Customer’. The Customer class also has a composition relationship with the Cart class which is depicted by a solid line and a filled diamond-shaped arrowhead pointing from the composite class (Customer) to the component class (Cart). The composite class (Customer) represents the whole concept and the component class (Cart) represents part of the whole. They have a composite relationship because if the Customer class is destroyed then the Cart class will be destroyed too. Each ‘Customer’ is composited with zero or one ‘Cart’ and each ‘Cart’ is composited with exactly one ‘Customer’. The ‘Order’ has ‘Product’. They have a composition relationship which is depicted by a solid line and a filled diamond-shaped arrowhead pointing from the composite class (Product) to the component class (Order). The composite class (Product) represents the whole concept and the component class (Order) represents part of the whole. They have a composition relationship because if the Product class is destroyed then the Order class will also be destroyed. Each ‘Product’ is composited with zero or more ‘Order’ and each ‘Order’ is composited with one or more ‘Product’. The ‘Category’ and ‘Product’ have an aggregation relationship which is depicted by a solid line with an open diamond-shaped arrowhead pointing from the aggregate class (Category) to the component class (Product). The aggregate class (Category) represents the whole concept and the component class (Product) represents part of the whole. They have an aggregation relationship as the component objects of ‘Product’ can exist independently although the aggregation class (Category) is destroyed. Each ‘Category’ is composited with any number of ‘Product’ and each ‘Product is aggregated with exactly one ‘Category’. Lastly, the ‘Business Owner’ manages the ‘Product’ and they have an association relationship which is depicted by the solid line. Each ‘Business Owner’ is associated with any number of ‘Product’ and each ‘Product’ is associated with exactly one ‘Business Owner’.

UML Diagram

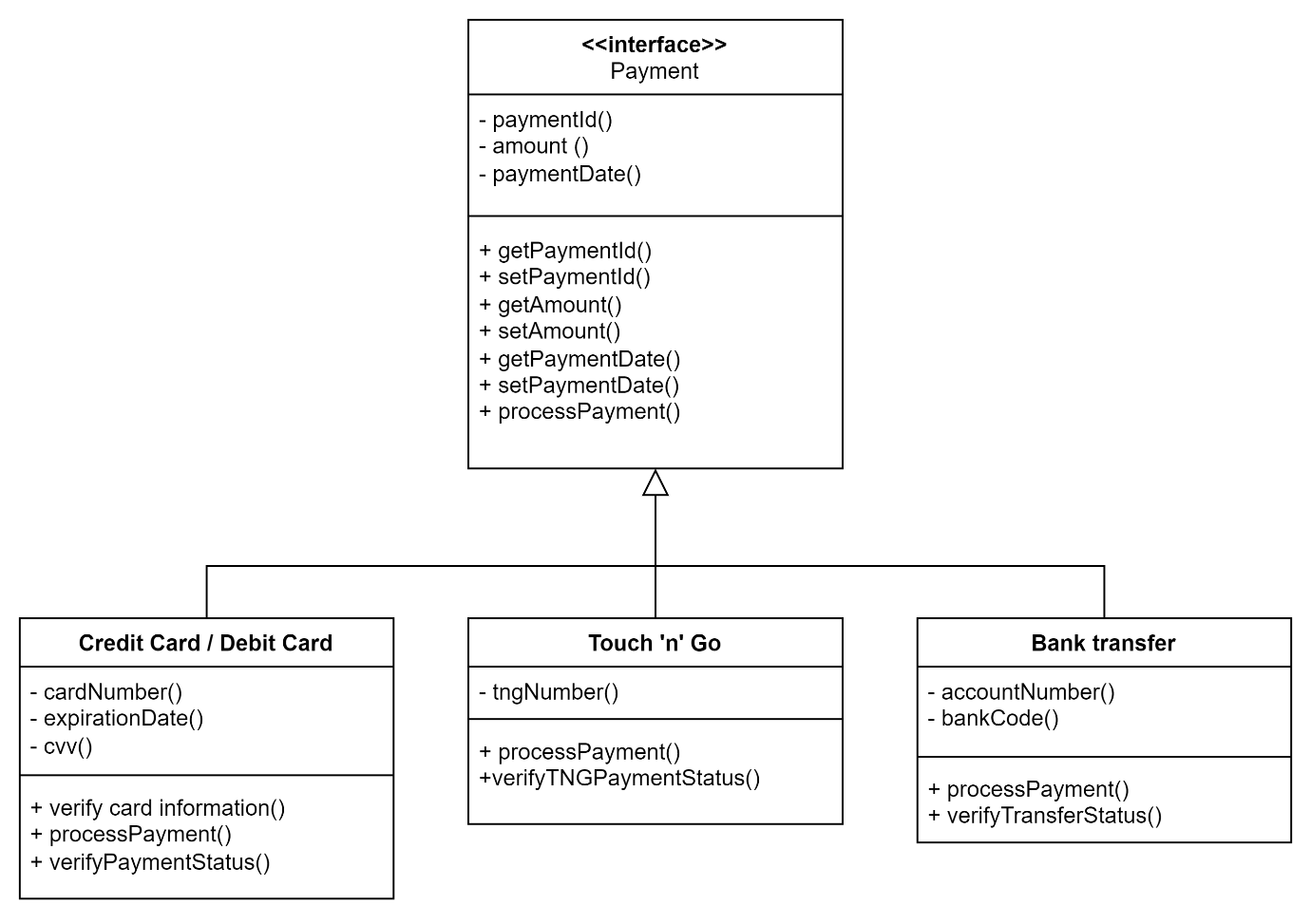


Figure 2

The problem is that there are no payment methods specified, so I have created a UML diagram (Figure 2) using the structural composite design pattern that has an interface ‘Payment’, which is the superclass associated with the payment methods: Credit Card/Debit Card, Touch 'n' Go and Bank transfer which are the three subclasses. The composite design pattern is a tree structure of simple and composite objects. I have chosen the structural composite design pattern for this problem as it allows to create a unified structure by treating individual objects and groups of objects uniformly. Other than that, it also provides flexibility and scalability in adding new elements to the composite structure and scalable design that accommodates various payment methods while providing a consistent interface for interacting with them. According to this design, the Payment interface (superclass) acts as a common contract for all payment methods which are Credit Card/Debit Card, Touch 'n' Go and Bank transfer. The payment methods which the subclasses represent the specific type of payments.

The Payment interface acts as a contract defining the common behaviour for different payment methods. It allows for treating different payment methods uniformly. It has payment(), amount() and paymentDate() as attributes and they have private visibility which is denoted by the ‘-‘ symbol that restricts direct access to them. The interface includes methods such as getPaymentId(), setPaymentId(), getAmount(), setAmount(), getPaymentDate(), setPaymentDate() and processPayment() which have public visibility denoted by the ‘+’ symbol indicating it can be used or accessed by other classes or components. The methods indicate getting and setting payment id, getting and setting amount, getting and setting payment date and finally processing the payment. The subclasses inherit the attributes and methods defined in the superclass. The Payment interface abstracts away the specific implementation details of payment processing. Credit Card/Debit Card which is the subclass represents a payment method using a credit card or a debit card. It has cardNumber(), expirationDate() and cvv() as the attributes which are visible privately denoted by the ‘-‘ symbol that restricts direct access to them. The attributes are the details needed to make the payment using a credit or debit card. It has verifycardinformation(), processPayment() and verifyPaymentStatus() as the publicly visible methods denoted by the ‘+’ symbol indicating that it is to be used or accessed by other classes or components. The methods indicate verifying card information and payment status and processing the payment. The Credit Card/Debit Card also implements the Payment interface. Next, the Touch ‘n’ Go subclass represents a payment method using Touch 'n Go. It has tngNumber() as the attribute which is the detail needed to make the payment with touch ‘n’ go and has private visibility denoted by the ‘-‘ symbol that restricts direct access to them. The methods include processPayment() and verifyTNGPaymentStatus() have public visibility denoted by the ‘+’ symbol indicating that they can be used or accessed by other classes or components. The methods indicate processing the payment and verifying the TNG payment status. It also implements the Payment interface. Lastly, the Bank transfer subclass has accountNumber() and bankCode() as the attributes which are the information needed to make the payment with bank transfer and they have private visibility denoted by the ‘-’ symbol that restricts direct access to them. It has processPayment() and verifyTransferStatus() as the methods that have public visibility denoted by the ‘+’ symbol indicating they can be used or accessed by other classes or components. The methods indicate processing the payment and verifying the transfer status. The Bank transfer also implements the Payment interface.