**ITECH1400 - Assignment 1 – Supermarket Self-Service Checkout**

**Student Name: Student ID:**

**Assignment Part 1 Details – Class Design**

***Insert your list/table of possible product properties here…***

**Product Properties (All)**

|  |  |  |  |
| --- | --- | --- | --- |
| barcode | name | price | quantity |
| weight | expensive | height | length |
| Discount available | purity |  |  |

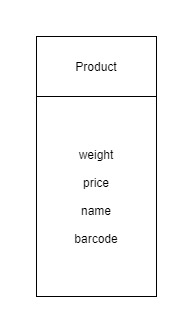
***Insert your list/table of key product properties here…***

**Product Properties (Key)**

|  |  |  |  |
| --- | --- | --- | --- |
| barcode | Price | weight | name |

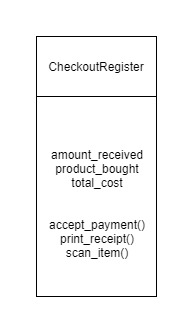
***Complete the class diagram of your final Product class here…***

**Product Class Diagram**



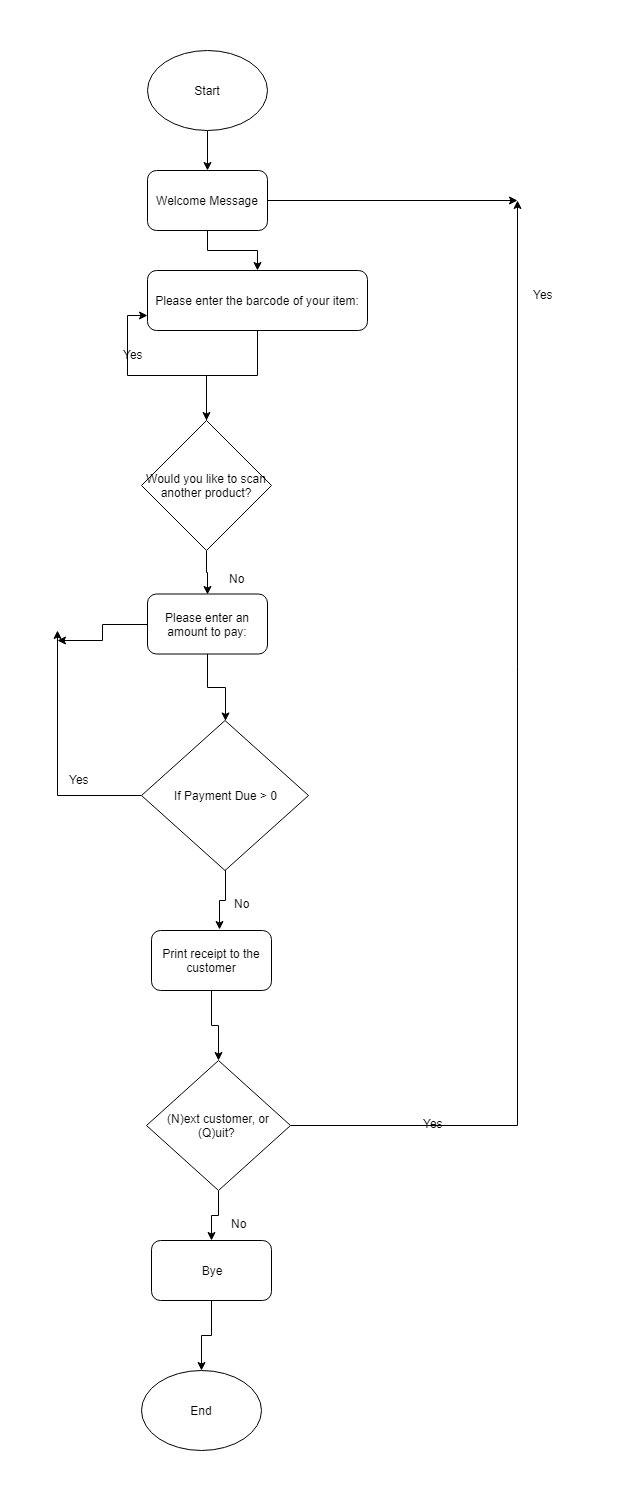
**CheckoutRegister Class Diagram**

***Complete the class diagram of your final CheckoutRegister class here…***



**Assignment Part 2 – Activity Flowchart**

***Insert your activity flowchart of the supermarket checkout process here… If your flowchart is large then place it on the following page.***

****

**Assignment Part 3 – Software Implementation**

Do not place your code here – provide the code as separate .py files submitted with this document.

**Assignment Part 4 – Code Explanation and Use**

***Update the below code to insert comments describing what the code is doing – for each line starting with a hash symbol (#) you should write your code comments after the hash. You may add a second line of comments if you require more space.***

**# Function to: get the** money **from the user**

**def** get\_float**(**prompt**):**

**# \_initializing the** money **to 0.0**

value **=** float**(**0.0**)**

**# keep asking the user until he enters the valid money**

**while** **True:**

**try:**

**# take input from the user and convert it to float**

value **=** float**(**input**(**prompt**))**

**# if user enters** negative money then display a message and ask again

**if** value **<** 0.0**:**

**print(**"We don't accept negative money!"**)**

**continue**

**# if user enters** valid money then break loop

**break**

**# if user enters** invalid money then display a message

**except** ValueError**:**

**print(**'Please enter a valid floating point value.'**)**

**# return the money entered by user**

**return** value

**# Function to: add products in bag**

**def** bag\_products**(**product\_list**):**

**# initialize bag and** non\_bagged\_items **to empty and** MAX\_BAG\_WEIGHT to 5.0

bag\_list **=** **[]**

non\_bagged\_items **=** **[]**

MAX\_BAG\_WEIGHT **=** 5.0

**# to iterate over** product\_list to get  **product**

**for** product **in** product\_list**:**

**# weight of product is more than 5.0 then remove this product**

**# from** product\_list and add the product to non\_bagged\_items

**if** product**.**weight **>** MAX\_BAG\_WEIGHT**:**

product\_list**.**remove**(**product**)**

non\_bagged\_items**.**append**(**product**)**

**# initialize the current bag contents to empty and current bag wt =0**

current\_bag\_contents **=** **[]**

current\_bag\_weight **=** 0.0

**# loop until there is no product in the product list**

**while** len**(**product\_list**)** **>** 0**:**

**# remove the product from the** product\_list located at index 0 #and store in temp\_product

temp\_product **=** product\_list**[**0**]**

product\_list**.**remove**(**temp\_product**)**

**# if sum of weight of current bag and** temp\_product is less than 5.0

**if** current\_bag\_weight **+** temp\_product**.**weight **<** MAX\_BAG\_WEIGHT**:**

**# then add this product to current bag**

current\_bag\_contents**.**append**(**temp\_product**)**

current\_bag\_weight **+=** temp\_product**.**weight

**# if product list is empty then add** current\_bag\_contents to bag list

**if** **(**len**(**product\_list**)** **==** 0**):**

bag\_list**.**append**(**current\_bag\_contents**)**

**# if sum of weight of current bag and** temp\_product is not less than

# 5.0 **then add** current\_bag\_contents to bag list

**else:**

bag\_list**.**append**(**current\_bag\_contents**)**

**# initialize the current bag contents to empty and current #bag wt = 0.0**

current\_bag\_contents **=** **[]**

current\_bag\_weight **=** 0.0

**# outputs the bag and index contents of the bag\_list**

**for** index**,** bag **in** enumerate**(**bag\_list**):**

output **=** 'Bag ' **+** str**(**index **+** 1**)** **+** ' contains: '

**# iterate over each product in bag**

**for** product **in** bag**:**

output **+=** product**.**name **+** '\t'

**print(**output**,** '\n'**)**

**# if there is any non bagged item then output them**

**if** **(**len**(**non\_bagged\_items**)** **>** 0**):**

output **=** 'Non-bagged items: '

**# iterate over each item in non bagged items**

**for** item **in** non\_bagged\_items**:**

output **+=** item **+** '\t'

**print(**output**,**'\n'**)**

**Assignment 1 – FedUni Checkout**

**Student name:**  **Student ID:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Part** | **Assessment Criteria** | **Weight** | **Mark** |
| 1a | Identification of properties of a typical supermarket **Product**. | 10 \* 0.5 = 5 marks |  |
| 1b | Application of abstraction to identify key properties of a typical supermarket **Product** as well as creation of a suitable **Class Diagram**. | 4 marks |  |
| 1c | Identification of the key properties of a **CheckoutRegister** as well as creation of a suitable **Class Diagram** which uses those properties, plus the four method signatures provided. | 4 marks |  |
| 2 | Creation of an **activity flowchart** which clearly indicates how the program should operate, using the correct symbols for elements such as start/end points, processes and decisions/branches | 10 marks |  |
| 3 | Programming of the product checkout simulation so that it:   * Creates a small number of **Product** instances that may be purchased, * Accepts simulated ‘scanning’ of a **Product** to identify it (including refusal to identify products which do not exist), * Adds a scanned **Product** to the **CheckoutRegister**’s list of products being purchased, * Allows the checkout of multiple products, * Accepts ‘virtual money’ to pay for those products (you must pay enough to cover the cost of the items checked out), and * Prints a final receipt of the products purchased, along with the total cost, total paid and any change given. | 5 + 5 + 5 + 5 + 5 + 5 = 30 marks. | i)  ii)  iiI)  iv)  v)  vi)  Total: |
| 4a | Analysis and documentation via code comments of the two functions provided. | (8 \* 0.5) + (16 \* 0.5) = 12 marks |  |
| 4b | Incorporation of the two functions provided into your main submission so that the program does not crash when an illegal money value is provided, and also virtually ‘bags up’ the products purchased. | 2 |  |
|  |  |  |  |
|  | Assignment total (out of 65 marks) |  |  |
|  | Contribution to grade (out of 20 marks) |  |  |

**Comments:**