Eric Duong

Description of the Project

The main purpose of the project was to create a simple shopping cart system for an e-commerce platform, which includes advanced object-oriented programming concepts. I essentially made a shopping cart which a user could then use to purchase items. Similar to walking into a store, putting things in your cart or removing them, then checking out. Using advanced object-oriented programming concepts such as Encapsulation, Polymorphism, Inheritance, and Abstraction the system looks more simplistic, although it is complicated to follow. The main classes of my program were “Product”, “Cart”, and “User.” There includes other classes but they are just addons or simply exist to be able to perform such advanced concepts like the ones listed previously. I used two different products, a PhysicalProduct and a DigitalProduct. Every class interacts with each other uniquely in order to perform complex tasks. Simply put, a User has a Cart with Products which they can freely use and checkout with Discounts.

Instructions

The program contains 9 classes but the main ones to focus on while using the program is DigitalProduct and PhysicalProduct which are subclasses that inherit the Product class, as well as the Cart and User class. To set up Products you must first create them, simply make a DigitalProduct or PhysicalProduct object, following the parameters given. Then an empty Cart must be made, which is then given to the User. To add a Product to the Cart you simply use the method add_to_cart(####), this adds the object/Product into the users Cart. To remove a Product from the users Cart you simply use the method remove_product(product_id), this removes the product from the users Cart. If the user chooses to update the quantity of any Product they may have they can use the update_quantity(#) method, which updates the quantity of Product that the user has in their Cart. To apply discounts to the User you must create a certain discount that you want, whether it be a PercentageDiscount or a FixedAmountDiscount. Either or a certain discount class is made which is then applied to the User using user1.discount, setting it equal to the newly created discount object. At anytime when the user is shopping they may view their cart using the .cart.view_cart() method, since they must access their cart in order to view their cart. Then when the user is ready they may checkout using the checkout() method which calculates the total cost in the Cart as well as applies discounts to the total price, furthermore clearing the cart at the end.

Class Product

Attributes: product_id, name, price, quantity

update_quantity method:
1 argument; the new_quantity equals the old quantity

get_product_info method:

no arguments; prints out attributes

Class DigitalProduct

Inherits: Product

Attributes: file_size, download_link,
product_id, name, price, quantity

get_product_info method:

no arguments; uses Product classes same
method using super(), prints out attributes

Class PhysicalProduct

Inherits: Product

Attributes: weight, dimensions, product_id, name, price, quantity

get_product_info method:

no arguments; uses Product classes same method using super(), prints out attributes

Class AbstractCart

Inherits ABC

Attributes: None

@abstractmethod
add_product method:
1 argument; pass

@abstractmethod
remove_product method:
1 argument; pass

@abstractmethod
view_cart method:
0 argument; pass

@abstractmethod
calculate_total method:
0 argument; pass

@abstractmethod
clearCart method:
0 argument; pass

Class Cart

Inherits Product and AbstractCart

Attributes: __cart_items (a list, private)

add_product method:
1 argument; appends product objects into __cart_items

remove_product method:

1 argument; matches up product_id with product objects in __cart_items and removes it

view_cart method:

0 argument; prints out all the Product objects in __cart_items using get_product_info() from Product class

calculate_total method:
0 argument; calculates and returns the total of all price attributes in __cart_items

clearCart method:

0 argument; removes all objects in the list that is __cart_items

Class Discount

Inherits: ABC

Attributes: None

apply_discount method

1 arguments; pass total _amount

Class PercentageDiscount

Inherits: Discount

Attributes: percentage

apply_discount method

1 arguments; returns total_amount minus the percentage discount of total_amount

Class FixedAmountDiscount

Inherits: Discount

Attributes: amount

apply_discount method

1 arguments; returns total_amount minus amount

Class User

Attributes: user_id, name, cart, discount

add_to_cart method:
1 argument; adds product from User's cart using Cart classes add_product method

remove_from_cart method:

1 arguments; removes product from User's cart using Cart classes remove_product(product_id) method

checkout method:

0 arguments; calculates total using Cart classes calculate_total method, in the case of Discount given then apply_discount method is used from Discount. The cart isn then cleared using clearCart method from Cart class

<u>Brief Summary on Developed Classes:</u>

Product Class: The Product class is the backbone of the program, essentially making the following classes **DigitalProduct** and **PhysicalProduct**. The Product class is really what starts off the attributes, as well as having the update_quantity method which sets the newly given quantity to the original quantity and a get_product_method which prints out the attributes. Product would be inherited down the other class products.

DigitalProduct Class: Inherits the Product class but adds on some unique attributes that only exist for this certain class, such as file_size and download_link. It uses super() in order to get the attributes in Product class. This class also has a get_product_info method and uses super() to get the same method from its parent but adds on unique print() statements due to it's unique attributes.

PhysicalProduct Class: Similar to DigitalProduct this class has unique attributes and inherits Product, weight and dimension. Using super() in a similar fashion this class merely adds onto Product in both attributes and method, like get_product_info.

AbstractCart Class: This class is simply made in order to make the future Cart class have abstract methods. Containing every method that the Cart class may have but making it an @abstractmethod.

Cart Class: This class is considered the most important class as it really holds the Product objects. Simply put, a private list attribute is created __cart_items which will hold all of the objects that the user may want in their cart. Using methods such as add_product, remove_product, view_cart, clearCart, and calculate_total, the user can interact with the objects while putting it into their cart. The names of the methods are pretty self-explanatory. But Cart has to inherit Product in order to use the objects that will go into the cart, as well as having access to their methods. So really it is a chain of inheritance.

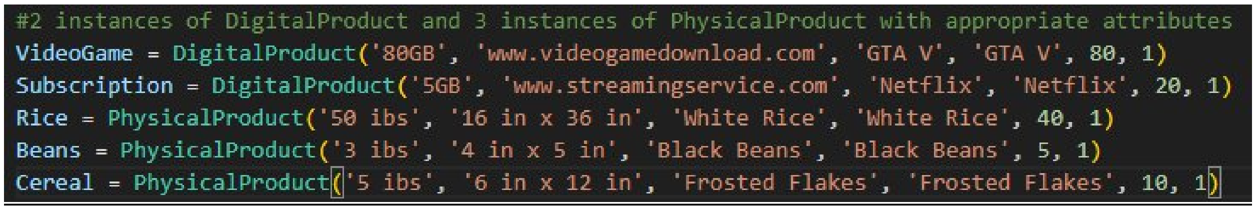
Discount Class: This class inherits ABC which similarly like AbstractCart, allows for the apply_discount method to be an @abtractemethod, for later classes. A prime example of Polymorphism.

PercentageDiscount Class: Inherits Discount in order to have the apply_discount class as abstract. With the given attribute of a percentage/decimal, it calculates and returns the cost/total that would be if the discount were applied.

FixedAmountDiscount Class: Very similar to the previous class this one basically is just a fixed amount discount, meaning it takes off integers (certain price instead percentage). But using the polymorphism concept, a way of putting this would be a PercentageDiscount or FixedAmountDiscount may be a Discount, but a Discount is not always a PercentageDiscount or a FixedAmountDiscount, it can be either or. But still taking on the method.

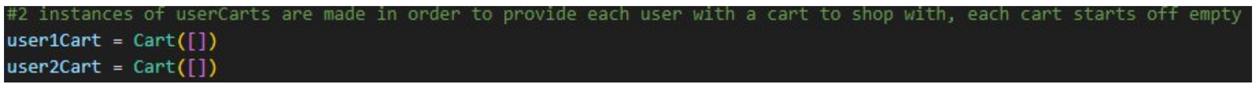
User Class: This class is where the magic really happens, where the program and classes get fully used together like a well oiled machine. Making an instance of cart and discount in order to use its methods the User class has methods that consist of add_to_cart, remove_from_cart, and checkout. All the methods use methods within the other classes. For example, remove_from_cart uses .remove_product from the Cart class, because it uses the cart instance attribute and therefore it has access to the methods in Cart. The checkout method is where the discounts are applied, the class knows when to use the discount because it would print out the original price otherwise if a discount was never given, if it was then it would go through an if statement and use Discount’s method apply_discount, then clear the cart using Cart’s clearCart method. A well oiled machine where everything works together.

<u>Verification of Sanity of Code:</u>



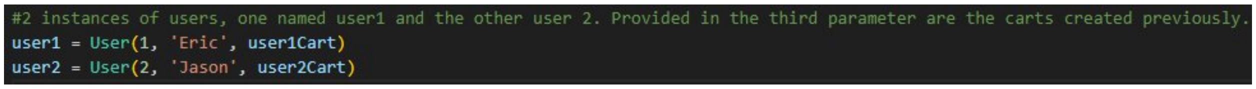
```
#2 instances of DigitalProduct and 3 instances of PhysicalProduct with appropriate attributes
VideoGame = DigitalProduct('80GB', 'www.videogamedownload.com', 'GTA V', 'GTA V', 80, 1)
Subscription = DigitalProduct('5GB', 'www.streamingservice.com', 'Netflix', 'Netflix', 20, 1)
Rice = PhysicalProduct('50 ibs', '16 in x 36 in', 'White Rice', 'White Rice', 40, 1)
Beans = PhysicalProduct('3 ibs', '4 in x 5 in', 'Black Beans', 'Black Beans', 5, 1)
Cereal = PhysicalProduct('5 ibs', '6 in x 12 in', 'Frosted Flakes', 'Frosted Flakes', 10, 1)
```

2 DigitalProduct objects are made as well as 3 PhysicalProduct objects, with appropriate attributes given the parameters.



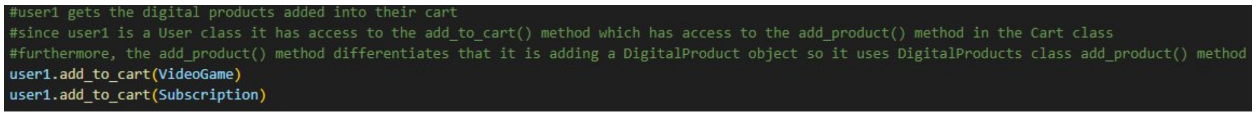
```
#2 instances of userCarts are made in order to provide each user with a cart to shop with, each cart starts off empty
user1Cart = Cart([])
user2Cart = Cart([])
```

Empty Cart classes must be given to the User class, if you went shopping you need a cart to shop.

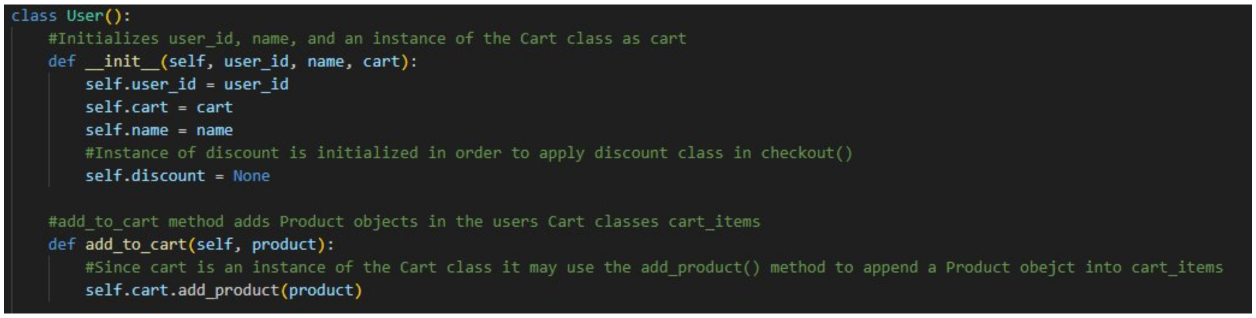


```
#2 instances of users, one named user1 and the other user 2. Provided in the third parameter are the carts created previously.
user1 = User(1, 'Eric', user1Cart)
user2 = User(2, 'Jason', user2Cart)
```

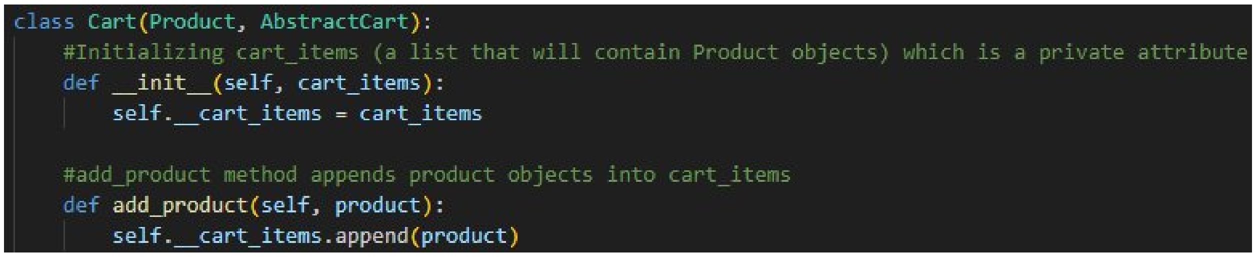
The Carts are given to the user



```
#user1 gets the digital products added into their cart
#since user1 is a User class it has access to the add_to_cart() method which has access to the add_product() method in the Cart class
#furthermore, the add_product() method differentiates that it is adding a DigitalProduct object so it uses DigitalProducts class add_product() method
user1.add_to_cart(VideoGame)
user1.add_to_cart(Subscription)
```

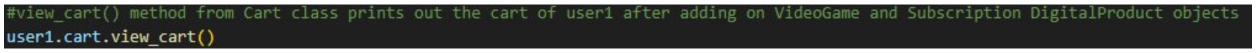


```python
class User():
    #Initializes user_id, name, and an instance of the Cart class as cart
    def __init__(self, user_id, name, cart):
        self.user_id = user_id
        self.cart = cart
        self.name = name
        #Instance of discount is initialized in order to apply discount class in checkout()
        self.discount = None

    #add_to_cart method adds Product objects in the users Cart classes cart_items
    def add_to_cart(self, product):
        #Since cart is an instance of the Cart class it may use the add_product() method to append a Product obejct into cart_items
        self.cart.add_product(product)
```

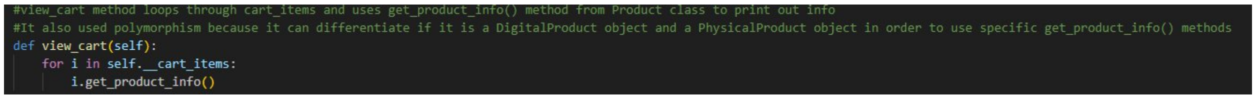


```python
class Cart(Product, AbstractCart):
    #Initializing cart_items (a list that will contain Product objects) which is a private attribute
    def __init__(self, cart_items):
        self.__cart_items = cart_items

    #add_product method appends product objects into cart_items
    def add_product(self, product):
        self.__cart_items.append(product)
```

A user1, which is a User object, is given the DigitalProduct objects VideoGame and Subscription which is a method from User which is used in the Cart class to append Product objects.

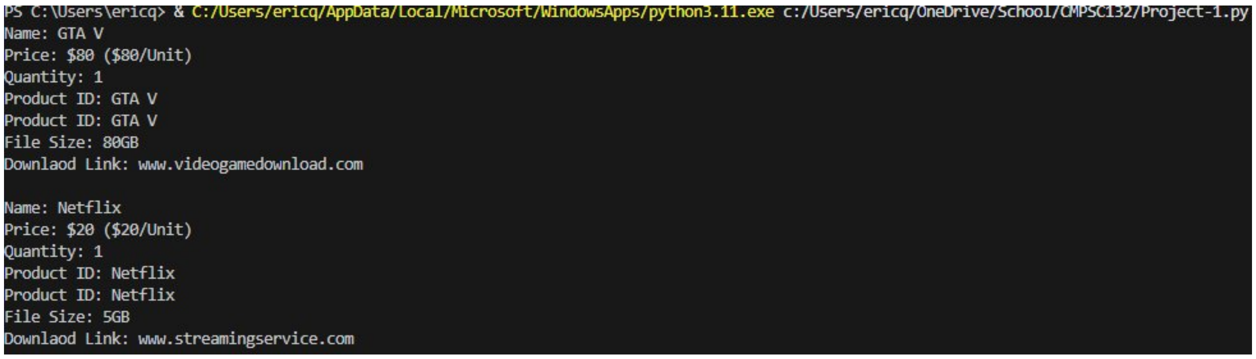


```python
#view_cart() method from Cart class prints out the cart of user1 after adding on VideoGame and Subscription DigitalProduct objects
user1.cart.view_cart()
```

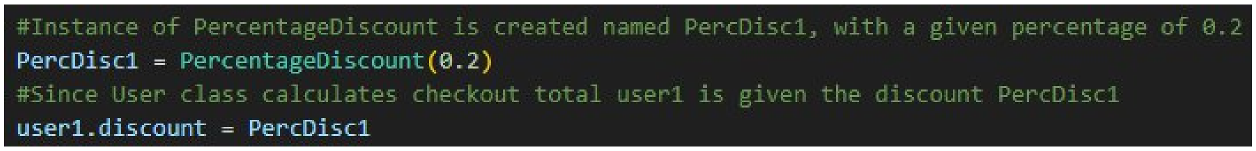


```python
#view_cart method loops through cart_items and uses get_product_info() method from Product class to print out info
#It also used polymorphism because it can differentiate if it is a DigitalProduct object and a PhysicalProduct object in order to use specific get_product_info() methods
def view_cart(self):
    for i in self.__cart_items:
        i.get_product_info()
```

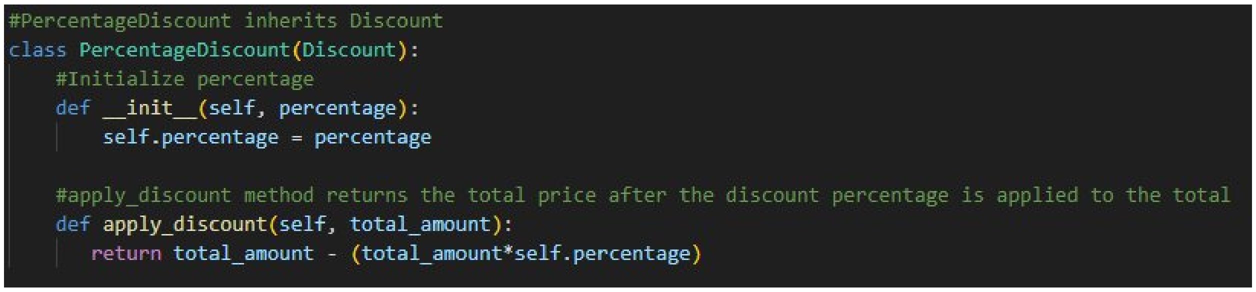
To test this we use the method view_cart() from the Cart class since user1 constains a Cart class object.



```
PS C:\Users\ericq> & C:/Users/ericq/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ericq/OneDrive/School/CMPSC132/Project-1.py
Name: GTA V
Price: $80 ($80/Unit)
Quantity: 1
Product ID: GTA V
Product ID: GTA V
File Size: 80GB
Downlaod Link: www.videogamedownload.com

Name: Netflix
Price: $20 ($20/Unit)
Quantity: 1
Product ID: Netflix
Product ID: Netflix
File Size: 5GB
Downlaod Link: www.streamingservice.com
```

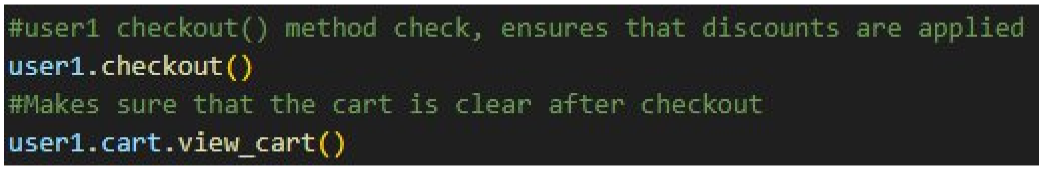


```python
#Instance of PercentageDiscount is created named PercDisc1, with a given percentage of 0.2
PercDisc1 = PercentageDiscount(0.2)
#Since User class calculates checkout total user1 is given the discount PercDisc1
user1.discount = PercDisc1
```

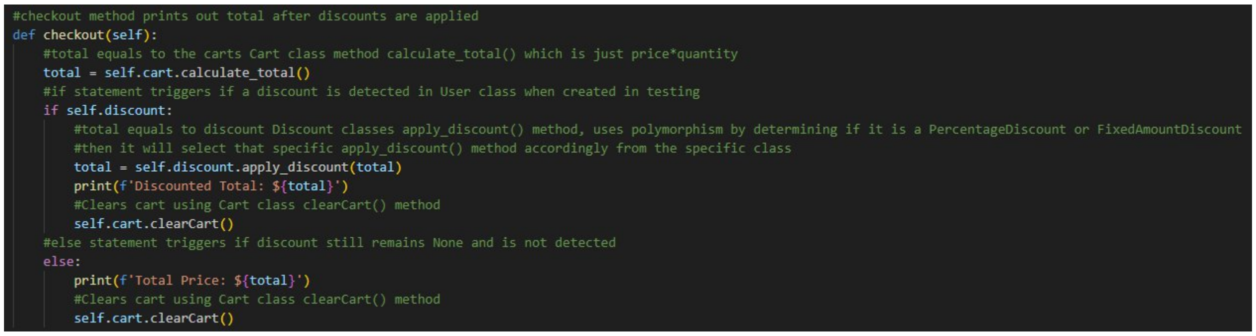


```
#PercentageDiscount inherits Discount
class PercentageDiscount(Discount):
    #Initialize percentage
    def __init__(self, percentage):
        self.percentage = percentage

    #apply_discount method returns the total price after the discount percentage is applied to the total
    def apply_discount(self, total_amount):
        return total_amount - (total_amount*self.percentage)
```

Testing out discount we make a PercentageDiscount object and give it to user1 in order to allow the checkout() method to detect a discount and compute the total cost with the discount.

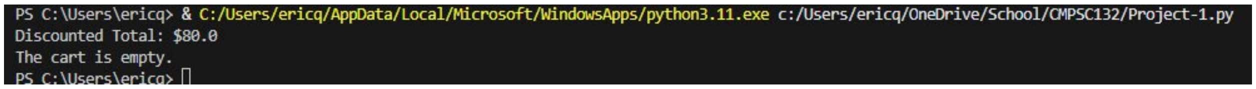


```
#user1 checkout() method check, ensures that discounts are applied
user1.checkout()
#Makes sure that the cart is clear after checkout
user1.cart.view_cart()
```

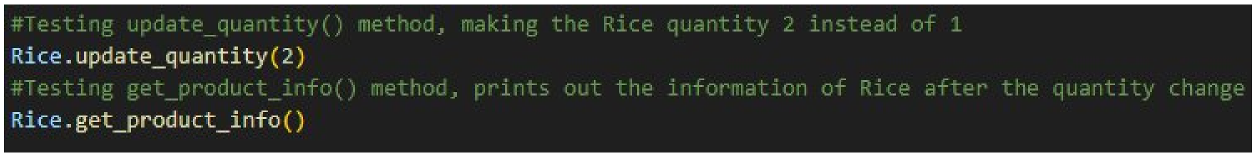


```
#checkout method prints out total after discounts are applied
def checkout(self):
    #total equals to the carts Cart class method calculate_total() which is just price*quantity
    total = self.cart.calculate_total()
    #if statement triggers if a discount is detected in User class when created in testing
    if self.discount:
        #total equals to discount Discount classes apply_discount() method, uses polymorphism by determining if it is a PercentageDiscount or FixedAmountDiscount
        #then it will select that specific apply_discount() method accordingly from the specific class
        total = self.discount.apply_discount(total)
        print(f'Discounted Total: ${total}')
        #Clears cart using Cart class clearCart() method
        self.cart.clearCart()
    #else statement triggers if discount still remains None and is not detected
    else:
        print(f'Total Price: ${total}')
        #Clears cart using Cart class clearCart() method
        self.cart.clearCart()
```

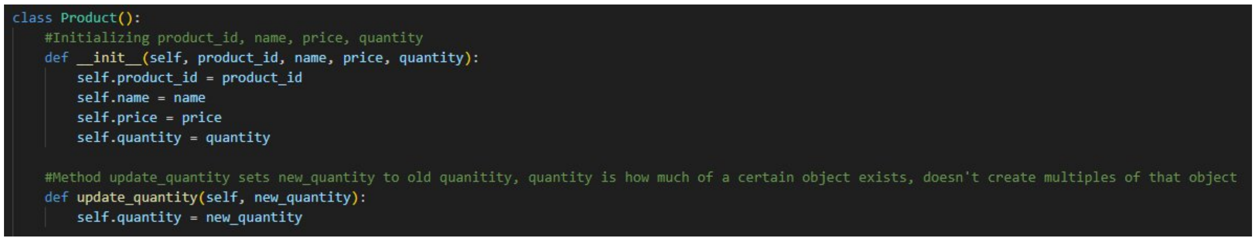
We use the checkout method and view_cart() method in order to verify if checkout really works.



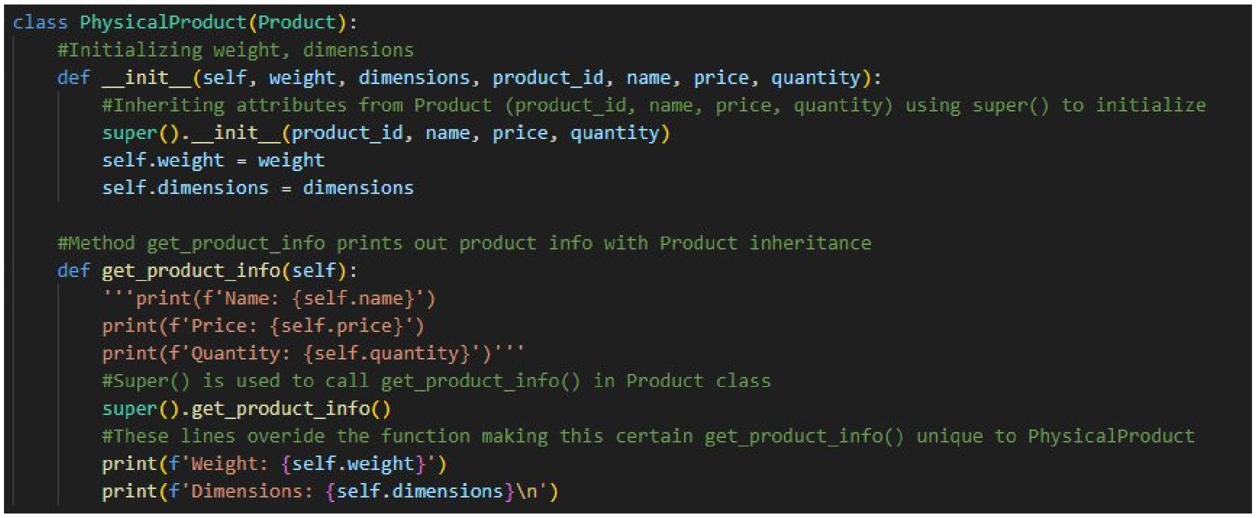
```
PS C:\Users\ericq> & C:/Users/ericq/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ericq/OneDrive/School/CMPSC132/Project-1.py
Discounted Total: $80.0
The cart is empty.
PS C:\Users\ericq> 
```



```
#Testing update_quantity() method, making the Rice quantity 2 instead of 1
Rice.update_quantity(2)
#Testing get_product_info() method, prints out the information of Rice after the quantity change
Rice.get_product_info()
```

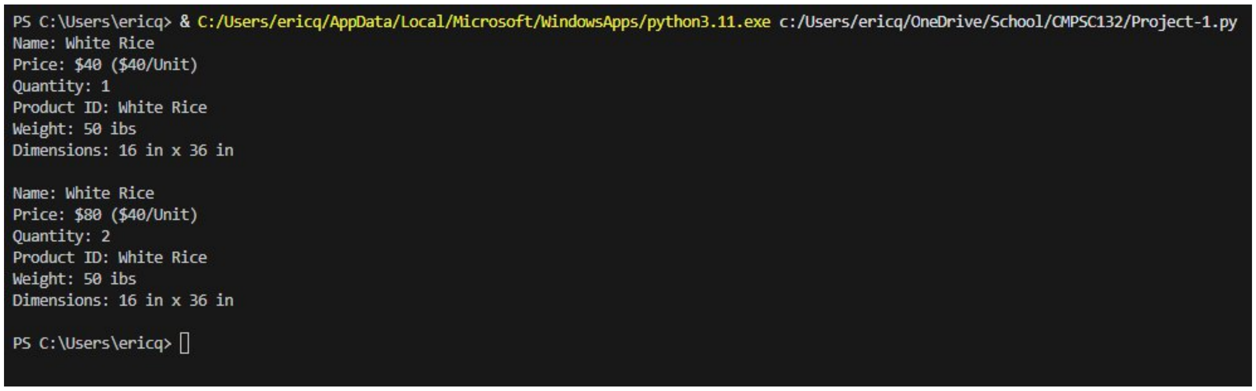


```
class Product():
    #Initializing product_id, name, price, quantity
    def __init__(self, product_id, name, price, quantity):
        self.product_id = product_id
        self.name = name
        self.price = price
        self.quantity = quantity

    #Method update_quantity sets new_quantity to old quanitity, quantity is how much of a certain object exists, doesn't create multiples of that object
    def update_quantity(self, new_quantity):
        self.quantity = new_quantity
```

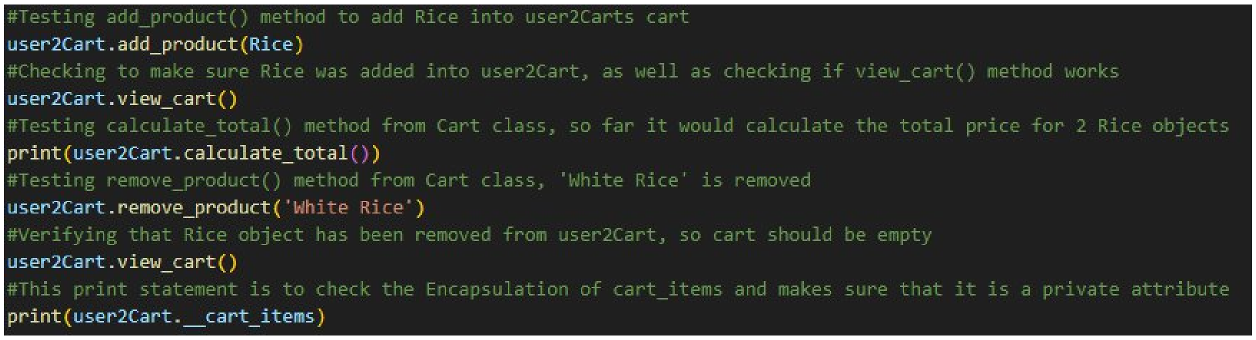


```python
class PhysicalProduct(Product):
    #Initializing weight, dimensions
    def __init__(self, weight, dimensions, product_id, name, price, quantity):
        #Inheriting attributes from Product (product_id, name, price, quantity) using super() to initialize
        super().__init__(product_id, name, price, quantity)
        self.weight = weight
        self.dimensions = dimensions

    #Method get_product_info prints out product info with Product inheritance
    def get_product_info(self):
        '''print(f'Name: {self.name}')
        print(f'Price: {self.price}')
        print(f'Quantity: {self.quantity}')'''
        #Super() is used to call get_product_info() in Product class
        super().get_product_info()
        #These lines overide the function making this certain get_product_info() unique to PhysicalProduct
        print(f'Weight: {self.weight}')
        print(f'Dimensions: {self.dimensions}\n')
```

We then test the update_quanity method by increasing the number of Rice by 2, verifying this by using the get_product_info method before and after, but specifically the PhysicalProduct's method because Rice is a PhysicalProduct.



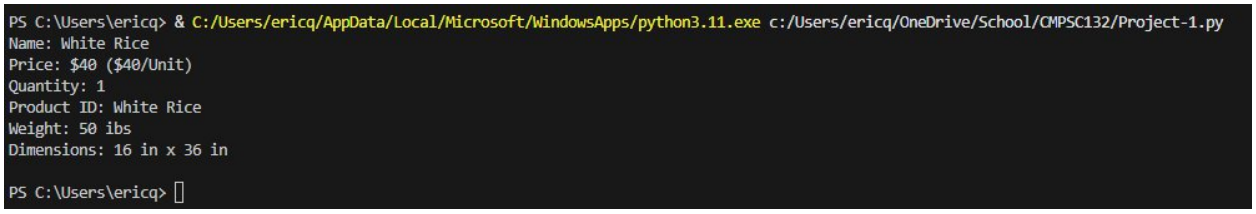
```
PS C:\Users\ericq> & C:/Users/ericq/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ericq/OneDrive/School/CMPSC132/Project-1.py
Name: White Rice
Price: $40 ($40/Unit)
Quantity: 1
Product ID: White Rice
Weight: 50 ibs
Dimensions: 16 in x 36 in

Name: White Rice
Price: $80 ($40/Unit)
Quantity: 2
Product ID: White Rice
Weight: 50 ibs
Dimensions: 16 in x 36 in

PS C:\Users\ericq> 
```



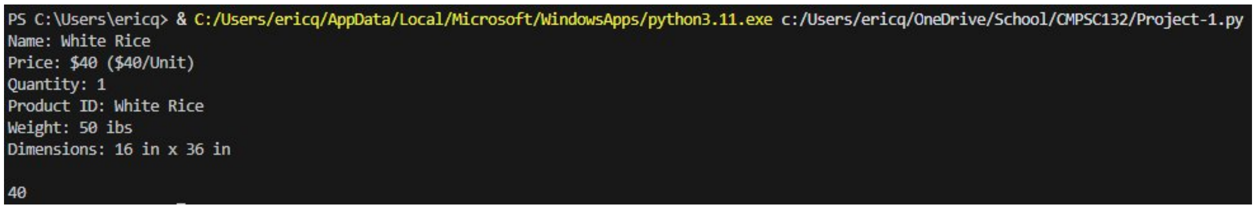
```python
#Testing add_product() method to add Rice into user2Carts cart
user2Cart.add_product(Rice)
#Checking to make sure Rice was added into user2Cart, as well as checking if view_cart() method works
user2Cart.view_cart()
#Testing calculate_total() method from Cart class, so far it would calculate the total price for 2 Rice objects
print(user2Cart.calculate_total())
#Testing remove_product() method from Cart class, 'White Rice' is removed
user2Cart.remove_product('White Rice')
#Verifying that Rice object has been removed from user2Cart, so cart should be empty
user2Cart.view_cart()
#This print statement is to check the Encapsulation of cart_items and makes sure that it is a private attribute
print(user2Cart.__cart_items)
```

We then check to make sure the methods within the Cart class work, by using each method then viewing the cart we see if the methods truly work. The bottom print statement prints out an error, this is a sanity check to make sure the __cart_items attribute truly is private.

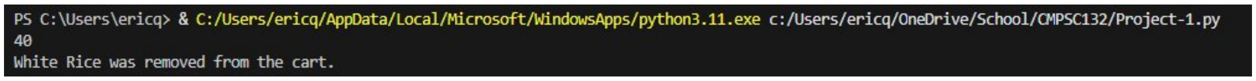
The results are as followed accordingly.



```
PS C:\Users\ericq> & C:/Users/ericq/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ericq/OneDrive/School/CMPSC132/Project-1.py
Name: White Rice
Price: $40 ($40/Unit)
Quantity: 1
Product ID: White Rice
Weight: 50 ibs
Dimensions: 16 in x 36 in

PS C:\Users\ericq> 
```

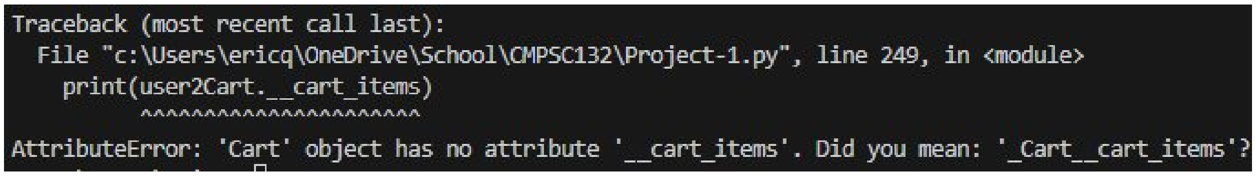


```
PS C:\Users\ericq> & C:/Users/ericq/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ericq/OneDrive/School/CMPSC132/Project-1.py
Name: White Rice
Price: $40 ($40/Unit)
Quantity: 1
Product ID: White Rice
Weight: 50 ibs
Dimensions: 16 in x 36 in

40
```



```
PS C:\Users\ericq> & C:/Users/ericq/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ericq/OneDrive/School/CMPSC132/Project-1.py
40
White Rice was removed from the cart.
```

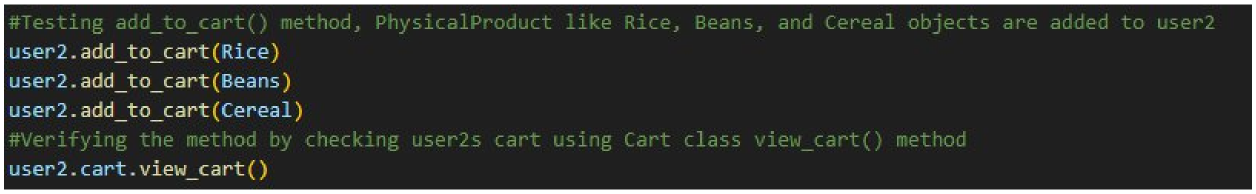
This is because Rice was the only thing in the cart.



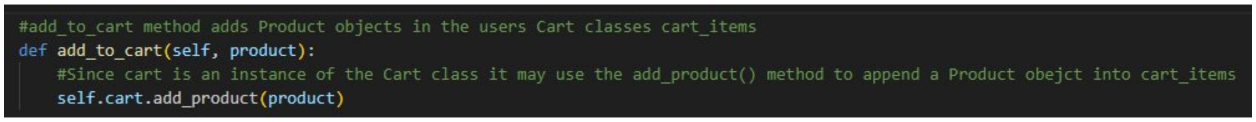
```
Traceback (most recent call last):
  File "c:\Users\ericq\OneDrive\School\CMPSC132\Project-1.py", line 249, in <module>
    print(user2Cart.__cart_items)
          ^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'Cart' object has no attribute '__cart_items'. Did you mean: '_Cart__cart_items'?
```

Sanity check for private attribute.

Now we test the User class.

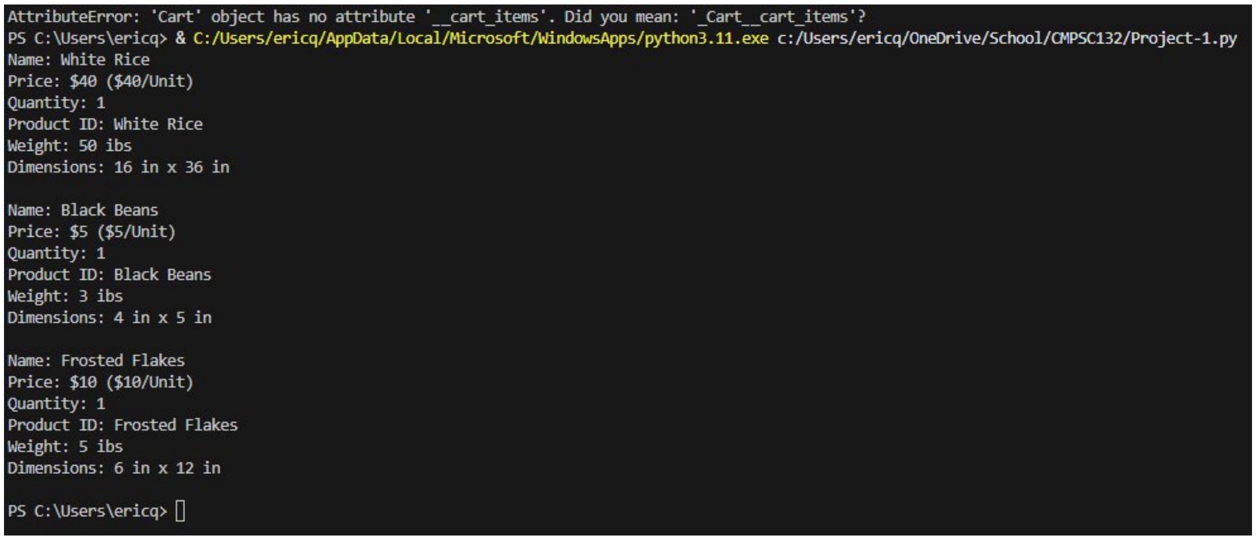


```
#Testing add_to_cart() method, PhysicalProduct like Rice, Beans, and Cereal objects are added to user2
user2.add_to_cart(Rice)
user2.add_to_cart(Beans)
user2.add_to_cart(Cereal)
#Verifying the method by checking user2s cart using Cart class view_cart() method
user2.cart.view_cart()
```

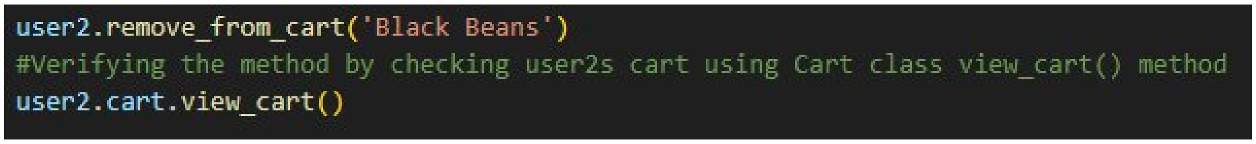


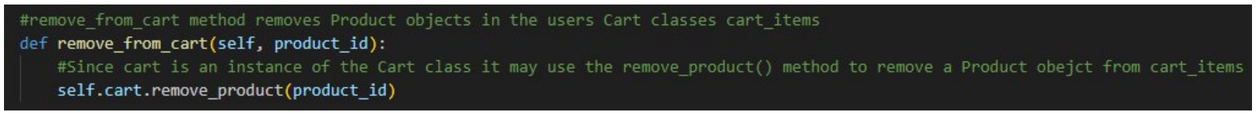
```
#add_to_cart method adds Product objects in the users Cart classes cart_items
def add_to_cart(self, product):
    #Since cart is an instance of the Cart class it may use the add_product() method to append a Product obejct into cart_items
    self.cart.add_product(product)
```

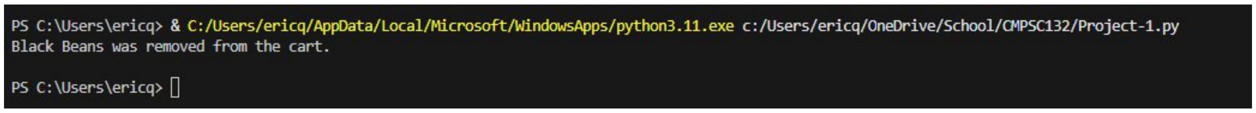
We add objects into user2 by using the add_to_cart method in User which uses Cart's method. We then verify by checking the cart.


```
AttributeError: 'Cart' object has no attribute '__cart_items'. Did you mean: '_Cart__cart_items'?
PS C:\Users\ericq> & C:/Users/ericq/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ericq/OneDrive/School/CMPSC132/Project-1.py
Name: White Rice
Price: $40 ($40/Unit)
Quantity: 1
Product ID: White Rice
Weight: 50 ibs
Dimensions: 16 in x 36 in

Name: Black Beans
Price: $5 ($5/Unit)
Quantity: 1
Product ID: Black Beans
Weight: 3 ibs
Dimensions: 4 in x 5 in

Name: Frosted Flakes
Price: $10 ($10/Unit)
Quantity: 1
Product ID: Frosted Flakes
Weight: 5 ibs
Dimensions: 6 in x 12 in

PS C:\Users\ericq> 
```

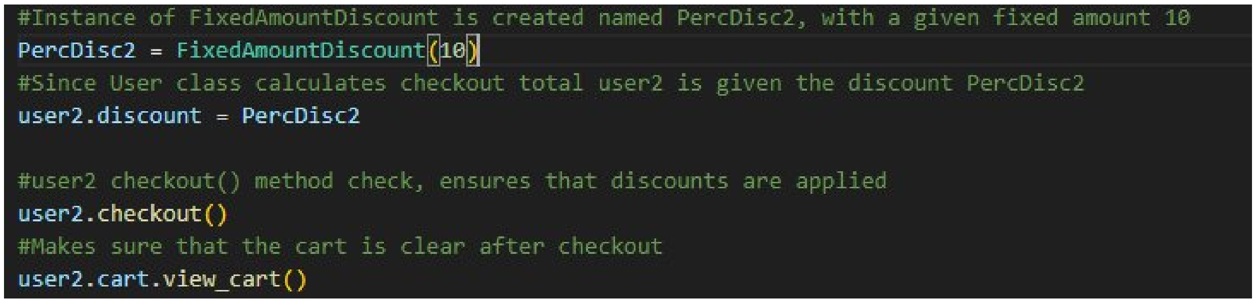
We then check the remove_from_cart method and verify with view_cart.

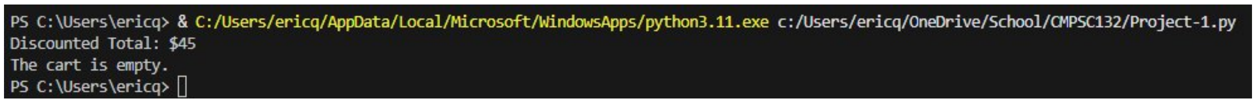

```
user2.remove_from_cart('Black Beans')
#Verifying the method by checking user2s cart using Cart class view_cart() method
user2.cart.view_cart()
```


```
#remove_from_cart method removes Product objects in the users Cart classes cart_items
def remove_from_cart(self, product_id):
    #Since cart is an instance of the Cart class it may use the remove_product() method to remove a Product obejct from cart_items
    self.cart.remove_product(product_id)
```

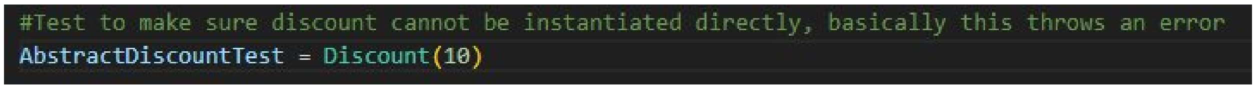

```
PS C:\Users\ericq> & C:/Users/ericq/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ericq/OneDrive/School/CMPSC132/Project-1.py
Black Beans was removed from the cart.

PS C:\Users\ericq> 
```

Discount is then checked along with the checkout method as done previously with user1.

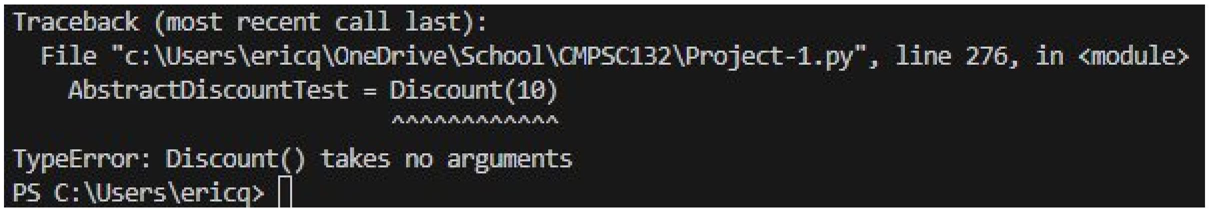

```
#Instance of FixedAmountDiscount is created named PercDisc2, with a given fixed amount 10
PercDisc2 = FixedAmountDiscount(10)
#Since User class calculates checkout total user2 is given the discount PercDisc2
user2.discount = PercDisc2

#user2 checkout() method check, ensures that discounts are applied
user2.checkout()
#Makes sure that the cart is clear after checkout
user2.cart.view_cart()
```


```
PS C:\Users\ericq> & C:/Users/ericq/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ericq/OneDrive/School/CMPSC132/Project-1.py
Discounted Total: $45
The cart is empty.
PS C:\Users\ericq> 
```

Finally we make sure discount cannot be instantiated directly.



```
#Test to make sure discount cannot be instantiated directly, basically this throws an error
AbstractDiscountTest = Discount(10)
```



```
Traceback (most recent call last):
  File "c:\Users\ericq\OneDrive\School\CMPSC132\Project-1.py", line 276, in <module>
    AbstractDiscountTest = Discount(10)
                           ^^^^^^^^^^^^
TypeError: Discount() takes no arguments
PS C:\Users\ericq> 
```

<u>Conclusion:</u>

I came across many issues during this project, like time management and making the coding work. But with the notes provided I was able to make everything connect. Making sure every object could have access to the methods that they were supposed to and that there were correct use of object-oriented programming concepts such as Inheritance, Polymorphism, Encapsulation, and Abstraction. Overall I found this project challenging but in a good way because I have already previously taken a similar course in High School. I look forward to more challenging questions/projects in the future as it will help me in my future job endeavors.