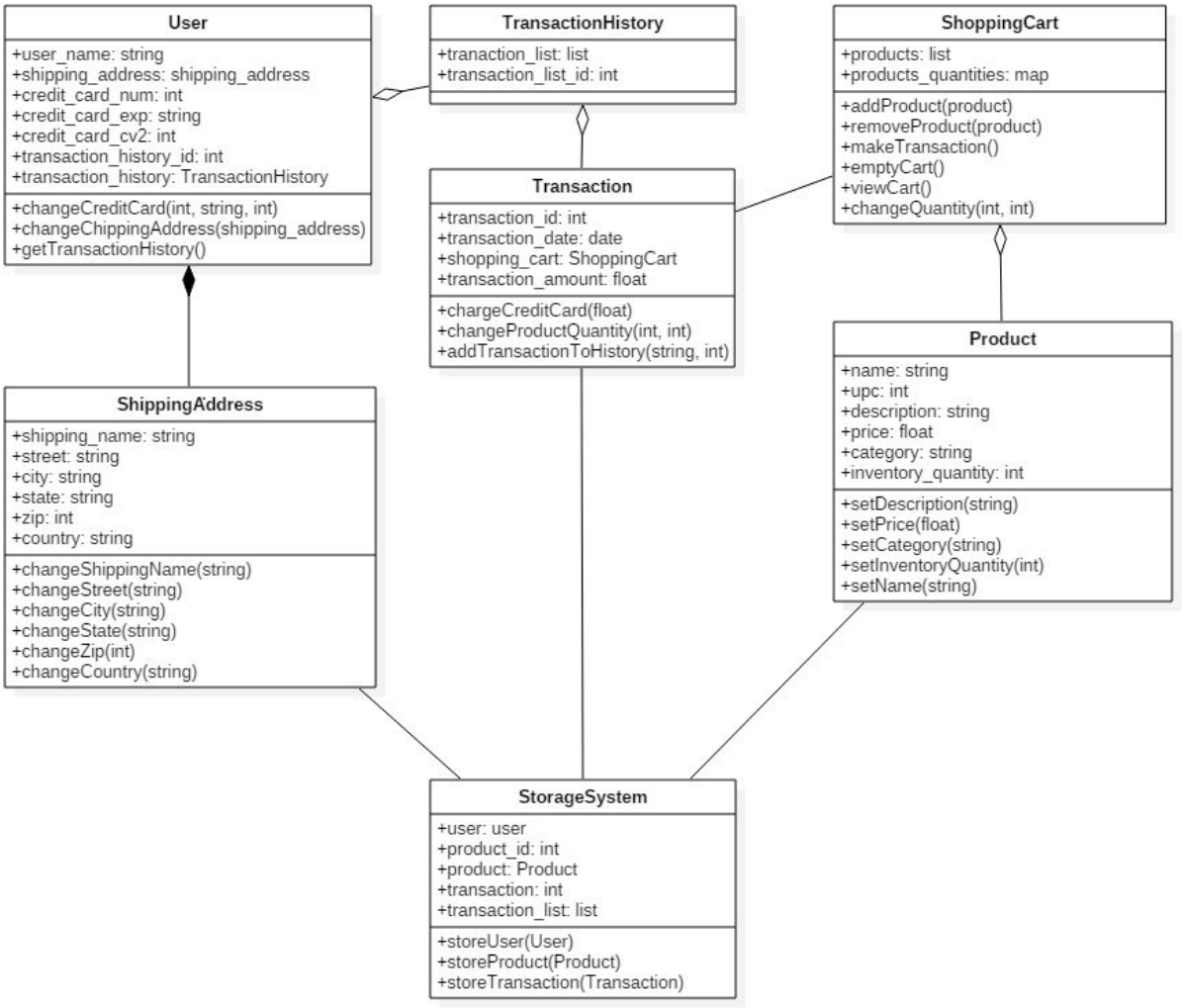
# In-Class Quiz / Assignment 2

August 30, 2016

Joseph Merrell-White (jam927)
Mauri Mitchell (mlm839)
Stephon Thornton (ust3)
Justin Wagner (jlw15)

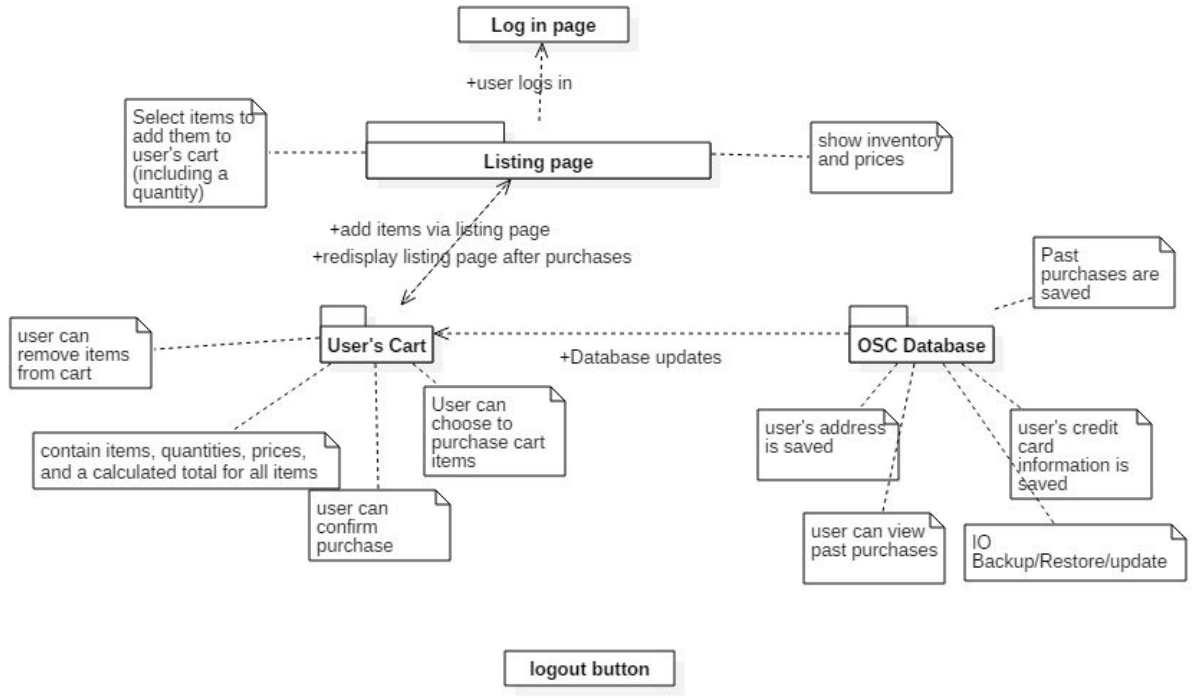
**Class diagram:**
The classes start with a User class, which contains all the information dealing with the user's profile, and with its composition class ShippingAddress, where ShippingAddress depends on User, and the ShippingAddress will cease to exist in the storage system if the user is deleted. The TransactionHistroy class stores the whole set of a users transactions, with Transaction being an aggregate class since each transaction will exist if the User and its TransactionList are deleted. Transaction stores information dealing with a users specific purchase. ShoppingCart stores the information about a users current product choices. It is aggregated with Product since the cart is built on top of a series of products. The Product class contains all the information on a specific product. The StorageSystem class is the interface between the classes and the physical medium in which the data is stored.

Two stakeholders who would benefit from this diagram would be database administrators and coders. Database administrators can use the diagram to decide the attributes and settings of the database, as well as the design of the tables, rows and keys. Coders can use the diagram to write the classes and their methods.



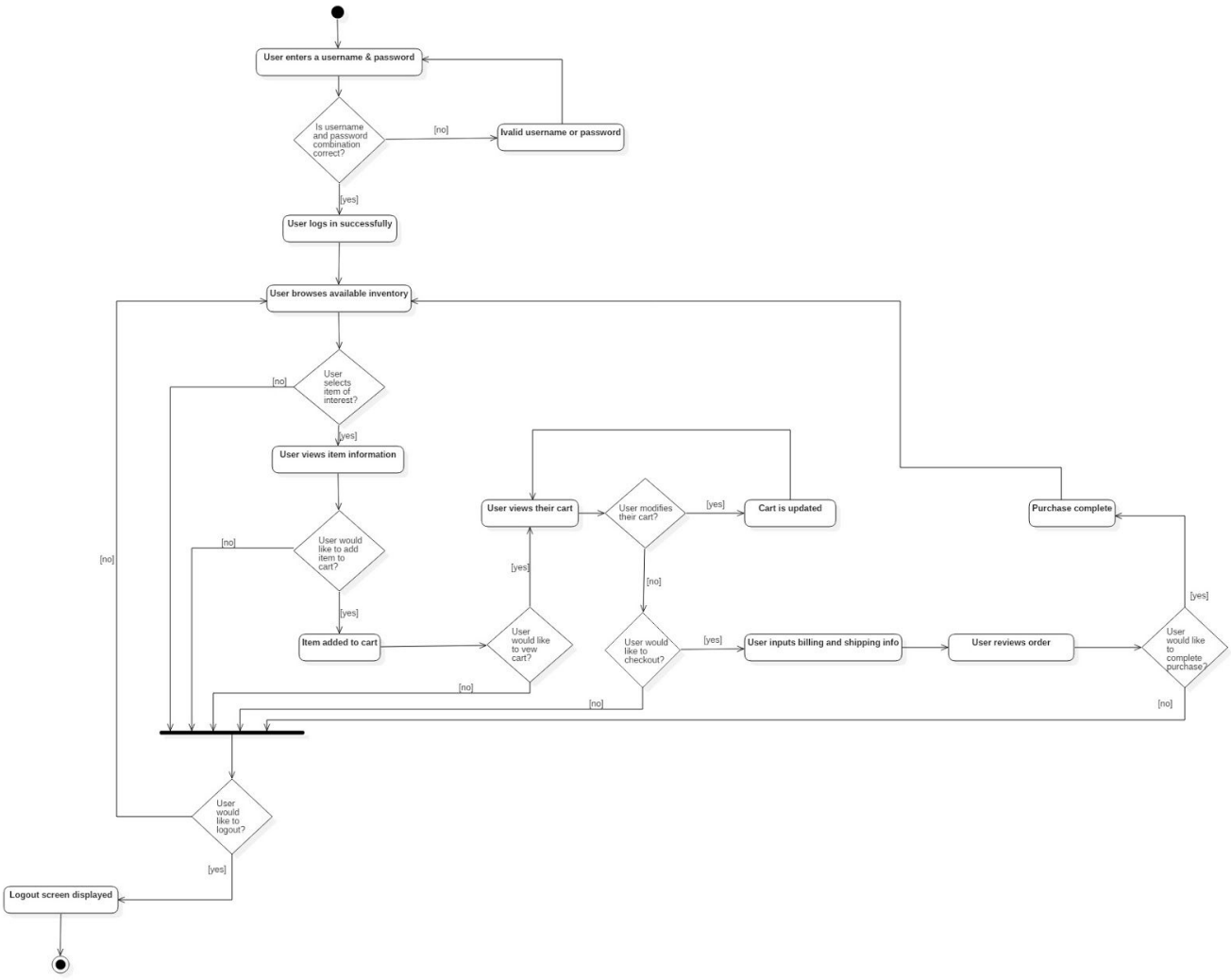
**Package Diagram**

The package diagram gives an overview of the online shopping center. It lists the different components of the OSC, and gives an overview of each component's functionality. The 2 stakeholders that would benefit from the package diagram is the software engineer and the business. The software engineer would benefit by knowing what should be included in the overall project. The business would benefit from this diagram because it would help them make sure all of their requirements have been provided.



**Activity Diagram**

The diagram below illustrates the process of a user logging in, creating a cart, modifying quantities of items, and making a purchase. The diagram contains Object-Oriented features such as: objects, classes, and abstraction. Stakeholders that may benefit from this diagram include software developers and users of the system. Software developers benefit from the diagram since it allows them to see a graphical workflow of how the system should work. Users may benefit from the diagram since it provides an easily comprehensible view of how the system should be used.

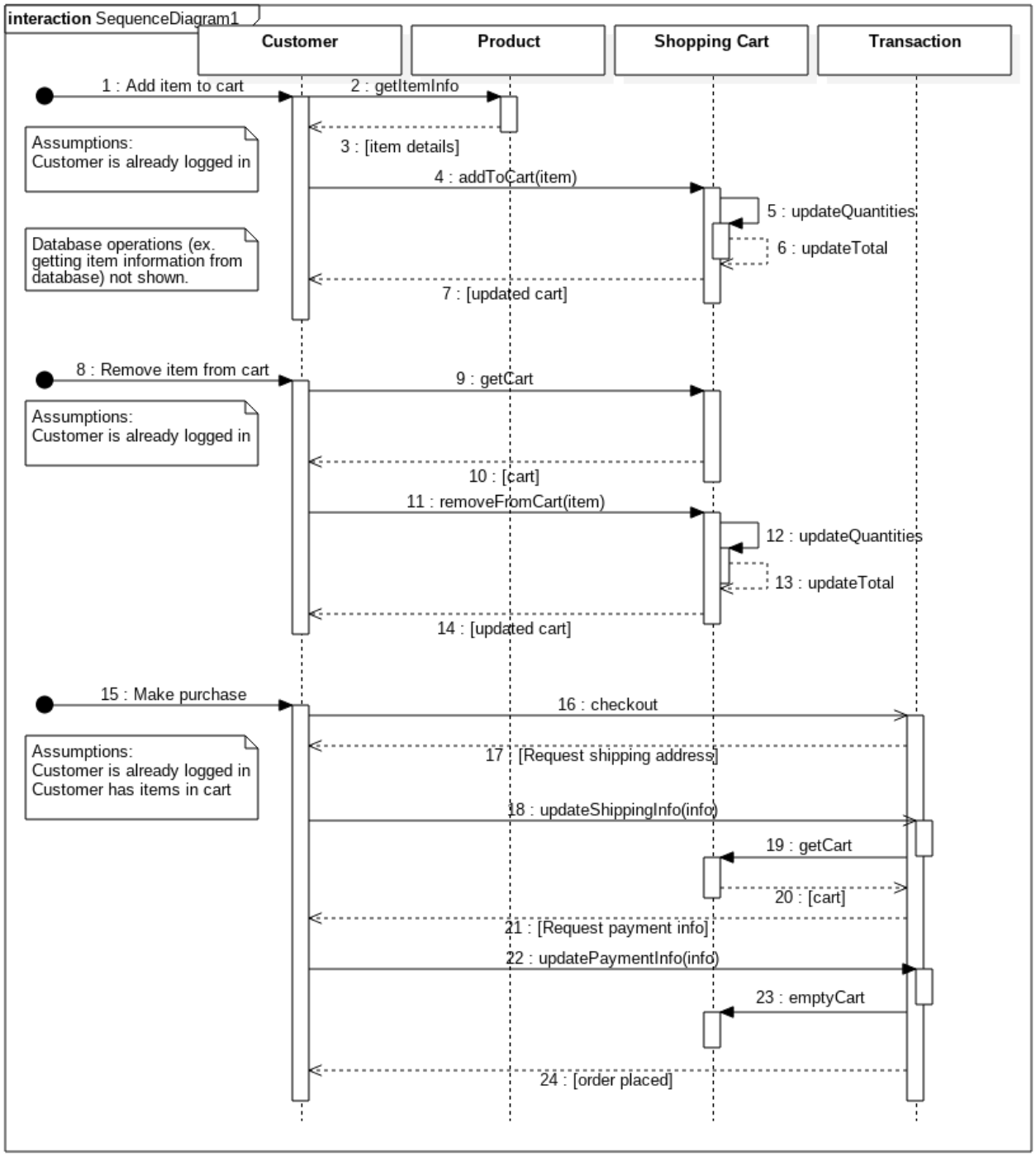


**Sequence Diagram**

The diagram following illustrates the sequence of events taken for adding or removing an item from the cart, and for placing an order with an existing cart. It shows a very high-level view of the interactions between the customer, product, shopping cart, and transaction class instances during these operations. Therefore, the diagram contains the following object-oriented features:

- Objects, as described immediately above
- Classes, in that these objects are instances of classes in the program
- Polymorphism, in that “Product” could refer to an instance of any subclass of the Product class
- Abstraction, in that this diagram is a very high-level view, and, as such, only presents the information needed to gain a fundamental understanding of the sequence of major events taking place during the illustrated operations
- Dependency, in that, in performing these operations, these classes require certain functionality from other classes

The developers of the program benefit from this diagram, as it gives them a basic structure with which to use as a guide when implementing the illustrated functionality. Security auditors may benefit from this diagram, as it gives an idea of the flow of potentially sensitive information.


interaction SequenceDiagram1
Customer
Product
Shopping Cart
Transaction
1 : Add item to cart
2 : getItemInfo
3 : [item details]
Assumptions:
Customer is already logged in
4 : addToCart(item)
5 : updateQuantities
6 : updateTotal
Database operations (ex. getting item information from database) not shown.
7 : [updated cart]
8 : Remove item from cart
9 : getCart
Assumptions:
Customer is already logged in
10 : [cart]
11 : removeFromCart(item)
12 : updateQuantities
13 : updateTotal
14 : [updated cart]
15 : Make purchase
16 : checkout
Assumptions:
Customer is already logged in
Customer has items in cart
17 : [Request shipping address]
18 : updateShippingInfo(info)
19 : getCart
20 : [cart]
21 : [Request payment info]
22 : updatePaymentInfo(info)
23 : emptyCart
24 : [order placed]

# Object-Oriented Practices

**Object**
An object is an instance of a class. Objects allow you to hold and manipulate certain data of a class.

**Class**
A class is a template that defines objects of the same type. Classes contain variables that define data fields and methods that define an object's behavior.

**Inheritance**
Inheritance is the ability of a new class to be created from an existing class by extending it. Inheritance allows a subclass to extend the functionality of a superclass by adding new types and methods or by overriding existing ones.

**Interface**
An Interface is a description of all functions that an object must have in order to be an instance of a particular class. It allows you to define the structure rather than the implementation, and it is useful when implementation may change frequently.

**Polymorphism**
Polymorphism is the ability to present the same interface for objects of different types. It allows you to have different classes with a function of the same name but different implementations.

**Abstraction**
Abstraction allows you to separate the use of a class from class implementation. It allows you to use a class like a black box without needing to know the complex details of the implementation.

**Encapsulation**
Encapsulation is the hiding of data implementation by restricting access to accessors and Mutators. Encapsulation allows you to make changes to a class without the worry that you will break other code that is calling or using the class.

**Modularity**
Modularity is the process of dividing a software system into multiple independent modules that are capable of carrying out tasks independently. Modularity allows designers and testers to execute or compile a piece of software separately from the entire system.

**Coupling**
Coupling is a measure that defines the level of dependability between modules of a program. It tells at what level the modules interfere and interact with each other. Low coupling means a better program design.

**Cohesion**
Cohesion is a measure that defines the degree of intra-dependability within elements of a module. High cohesion means a better program design.

**Aggregation**
Aggregation is similar to association. One class uses the other, however, the objects develop a part-whole relationship and the lifetime of part does not depend on lifetime of whole. If whole were to cease to exist, part would not be destroyed.

**Dependency**
Dependency can develop between objects while they are associated, aggregated or composed. This kind of relationship develops while one object invokes another object's functionality in order to accomplish some task. Any change in the called object may break the functionality of the caller.

**Composition**
In composition, classes also use each other, and there is also a part-whole relationship developed. However, in composition, the lifetime of part depends on the lifetime of whole. If whole were to cease to exist, so would part.

**Association**
Association is a reference based relationship between classes. One class holds a class level reference to the other class but neither of them is a part of the other.

**Object-Orientation -** What does it mean for a program to be object oriented?
For a program to be object-oriented, it must use a programming language model organized around objects and data rather than actions and logic. Object-oriented programming focuses on the objects we want to manipulate rather than the logic required to manipulate them and utilizes all of the principles defined above.

**Sources Used**
http://codebetter.com/raymondlewallen/2005/07/19/4-major-principles-of-object-oriented-programming/
(Defining Inheritance, Abstraction, Encapsulation, Polymorphism)

http://www.tutorialspoint.com/software_engineering/software_design_basics.htm
(Defining Modularity, Coupling, Cohesion)

http://www.codeproject.com/Articles/777540/Association-Aggregation-Composition-Dependency-and
(Defining Aggregation, Dependency, Composition, Association)

http://searchsoa.techtarget.com/definition/object-oriented-programming
(Defining Object-Orientation)