

Designing Software with an Object Oriented Perspective

Abstract: Present basic Object Oriented Software Design concepts at a level that is accessible to both experienced programmers and individuals that have not programmed

Keywords: Object Oriented, UML, Class Diagram, Object Diagram, State Diagram, Use Case Diagram, Sequence Diagram,

Author: Jason Ostroski

11/8/2013

Introduction

Object Oriented Software Design is a way designing programs utilizing models based on real-world ideas. Programming is often seen as how to design logic, but Object Oriented Programming focuses on the objects we want to manipulate more than the logic required to manipulate them. Designing software from an Object Oriented perspective allows programmers to communicate ideas, create more complex programs, and is one of the requirements in most software engineering job postings. These concepts are not specific to one programming language, but rather can be implemented in many different languages.

Objective

Teach the concepts of designing a program, rather than writing logic functions, and explain Object Oriented Software Design at a level assessable to both experienced programmers and people who have never programmed before.

(UML Standards utilized for all diagrams)

Key Concepts

There are four key concepts that form the base of Object Oriented Software Design. The first concept, identity, breaks up data into discrete parts, called objects. Objects can be concrete, such as a file or picture, or conceptual, such as a calculation. Every object has it's own identity, and two objects are their own entity even if all of their attributes are identical. The second concept, classification, groups objects with the same attributes and operations into a class. A class describes the important properties of an object. Each object is an instance of a class.

Example: A bicycle class would have attributes such as wheel size, color, number of gears and operations such as brake and shift gears. Any object of the bicycle class would be one specific bicycle that could have 24-inch wheels, red paint, 5 gears, and could brake quickly and shift gears slowly.

The specific bicycle object in the above example has its own value for each attribute, but shares the same attribute names and operations as other objects of the same bicycle class. The bicycle in the above example has 24-inch wheels and another bike could have 20-inch wheels, but both bikes have an attribute of wheel size. An object has knowledge of its class; therefore, the bicycle knows it is a bicycle.

The third concept, inheritance, is sharing attributes and operations among classes based on their relationships in a hierarchy. The top of the hierarchy is a superclass, which holds the general information, and the lower subclasses hold more specific information. Each subclass inherits the features of the superclass, but does not repeat these features.

Example: A superclass may be a bicycle with subclasses of road bicycles and mountain bicycles. The superclass, bicycle, could hold information for tires. The road bicycle subclass and mountain bicycle subclass would both hold their own information about their tire specifications. Both bicycles have tires, but they are different entities as the road bike tires have flat treads and the mountain bike tires have deep treads.

The fourth concept, polymorphism, is the idea that the same operation may behave in different ways for different classes. An operation is a procedure that an object performs or has performed to it. An operation of a specific class is a method.

Example: The move operations performed by different chess pieces are polymorphic. The move operation for a pawn behaves differently than the move operation for a queen, but both are move operations of chess pieces.

Modeling

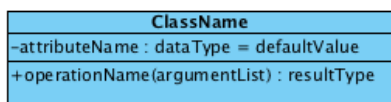
Three key types of models are utilized when designing a software system. Class Models describe the structure of objects in the system and include the objects' identities, relationships to other objects, attributes, and operations. State Models describe the aspects of objects that require time or sequencing and the events that cause the changing of states. Interaction Models describe how objects interact with each other and how

individual objects collaborate to make a system function as a whole. While each type of model describes different functionality, they all are necessary in realizing the system.

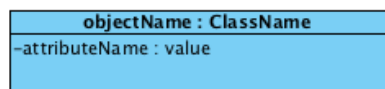
Class Modeling

Class Diagrams graphically represent modeling for classes and relationships, while Object Diagrams perform the same function for objects and are used to illustrate how an object is an instantiation of a class. Attributes and operations of classes are included in Class Diagrams and values of objects are included in Object Diagrams.

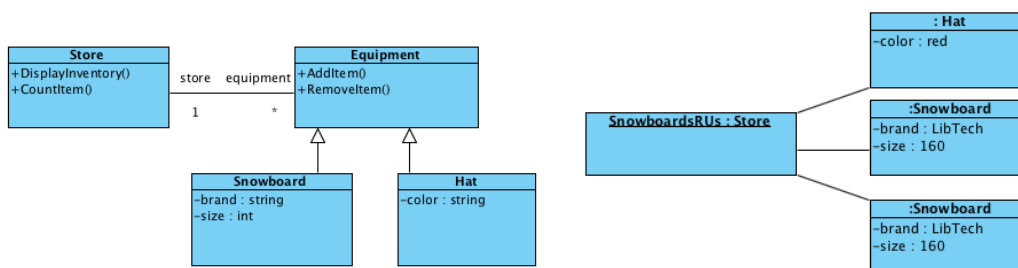
Class Diagram Format:



Object Diagram Format:



Example: A snowboard shop would like to keep track of their equipment with an inventory system. The store would like to be able to count and display items. The store current has snowboards (brands and sizes), and hats (colors). Design class and object diagrams of the software for the inventory system.



This solution is not unique for this problem, but it implements many important concepts of class and state diagrams. The Class Diagram on the left contains a store, which has operations for the system. The equipment class also has operations for adding an item and removing an item. The line connecting store and equipment is an association. The “store 1” and “equipment *” signifies the multiplicity of those classes. There is one

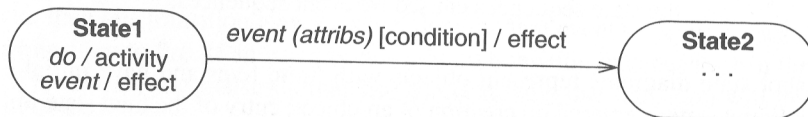
store for every piece of equipment, but there can be many pieces of equipment for that one store. The snowboard and hat classes have attributes and the arrow lines from these classes to the equipment class signify that they are subclasses of the equipment super class. The object diagram on the right shows the single instance of the store class as the SnowboardsRUs object. The connecting hat and snowboard classes have values for the attributes of their class counterparts. The line between the store and the other objects signify a link between the objects. There is no equipment object because the hat and snowboard subclasses are inherited from the equipment superclass and they are the instantiated objects from these super and subclasses.

Drawing class diagrams can simplify designing software classes and are a good reference while programming as they are a quick way to look up which classes contain which attributes and operations. Similarly, object diagrams easily show the objects that are created when the program runs.

State Modeling

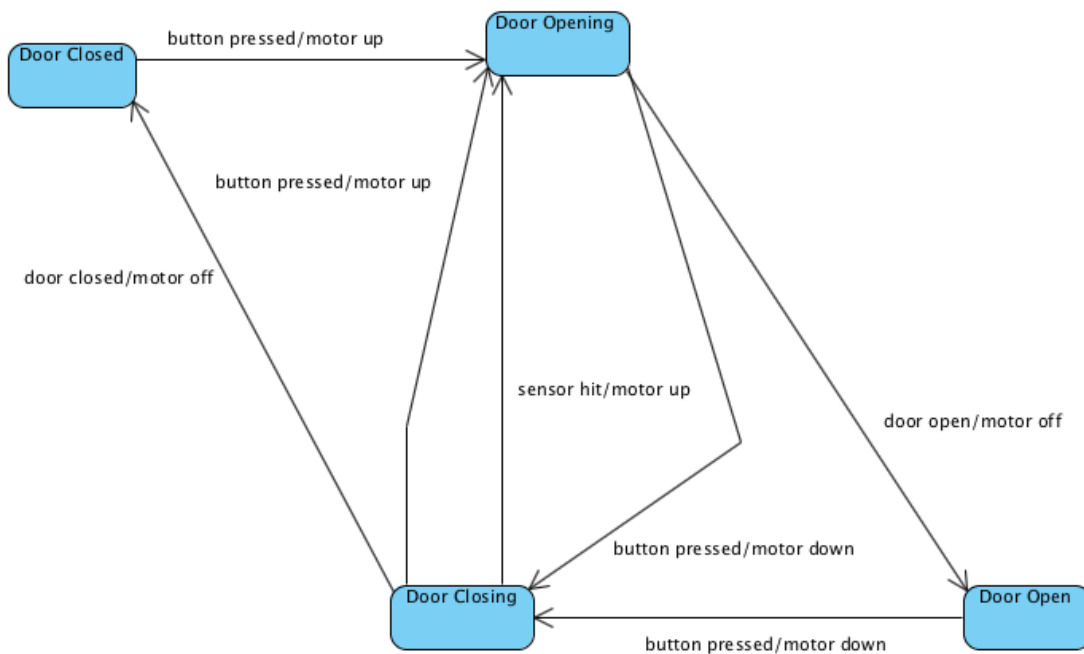
State Diagrams are graphs whose nodes are states and connections are transitions between states. States are abstractions of the values and links of an object, and sets of these values and links are grouped together into a state according to behaviors of the object. An event is when something occurs at a point in time.

State Diagram Format:



(Figure: Object Oriented Modeling and Designing with UML)

Example: Design a State Diagram for a garage door.



This is not a unique solution to this problem. When in the door closed state, if the button is pressed, the motor will pull the door up. Similarly, when in the door open state, if the button is pressed, the motor will go down. When in the door closing state, if the door reaches the ground, the door is now closed and the motor turns off. Similarly, when in the door opening state, if the door reaches the top, the door is now open and the motor turns off. Many garage doors allow you to hit the button while the door is in motion and it will move the opposite way, which is handled in two of the transitions between the door closing and door opening states. Many garage doors also have a sensor to ensure the door does not close on an object, therefore, if the sensor is tripped when door is closing, the door will change directions and move up.

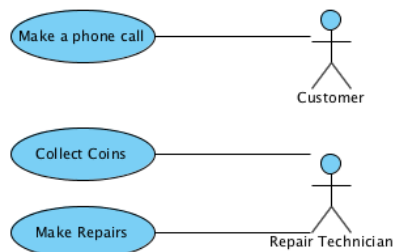
State Diagrams allow complex state changes to be realized and often make unrealized behaviors rise to attention when designing a program.

Interaction Modeling

Interaction diagrams are the final model in designing a system and describe how objects interact. Often an actor is used as an external object of the system that

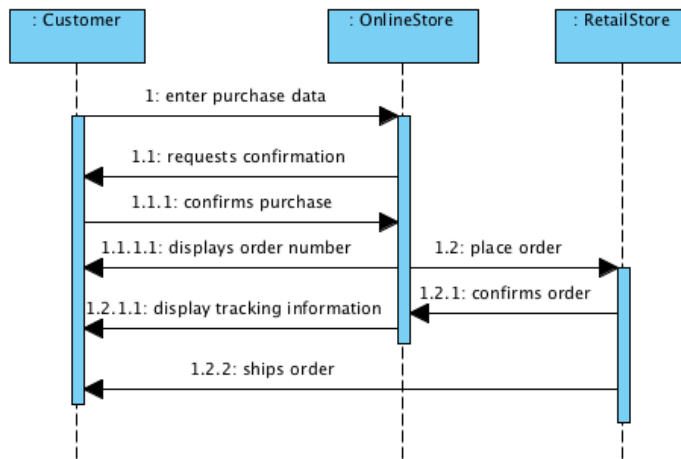
communicates with the system. A use case is a piece of functionality that the system can provide by interacting with an actor.

Example: A person actor could make a phone call on a pay phone. The act of making the phone call would be a use case. Another actor, a repair technician, could empty the coin collector or make repairs. Emptying the coin collector and making repairs are use cases and they are shown below in the use case diagram.



For more complex programs with sequences of actions, a sequence diagram must be utilized. A sequence diagram shows not only an actor and an interaction, but also the sequence of messages between all the interactions.

Example: A customer is trying to purchase an item from an online store that has a retail store for all of its stock. Create a sequence diagram for the interaction between the customer, online store, and retail store.



This solution is not unique. The customer first enters the purchase information. This starts the customer's lifeline, a vertical blue line that represents how long/when the

actor is active in the diagram. The online store will request a confirmation from the customer in order to validate the item selection, banking information, and shipping information. Once the customer confirms the purchase, the store will display the order number to the customer and place the order to the retail store at approximately the same time. The retail store confirms the order has been placed with the online store, which informs the customer of the tracking information. This ends the life cycle of the online store, as it is no longer needed in the sequence. The retail store then ships the order to the customer and the sequence is complete.

Sequence diagrams are very useful in plotting out the interactions of the system with approximate timing. Sequence diagrams can be much more detailed than use case diagrams, therefore, are utilized more often in complex design problems.

Conclusion and Recommended Resources

Each type of modeling is useful for different aspects of software design. They can be utilized independently for smaller, less complex problems, or in collaboration to solve highly detailed, complex problems. All the diagrams, with the exception of the State Diagram Format, were created utilizing Visual Paradigm for UML Community Edition, which is free for non-commercial usage. All of the Object Oriented concepts are covered in Dr. Charles B. Owen's, CSE 335 - Object Oriented Programming course offered at Michigan State University. For more information on designing Object Oriented software, consider reading Object Oriented Modeling and Design with UML.

References

Rouse, Margaret. "Object-Oriented Programming." SearchSOA. N.p., Aug. 2008. Web. 06 Nov. 2013. <<http://searchsoa.techtarget.com/definition/object-oriented-programming>>.

Blaha, Michael, and James Rumbaugh. Object-oriented Modeling and Design with UML. 2nd ed. Upper Saddle River, NJ: Pearson Education, 2005. Print.

Dr. Charles B. Owen's MSU Website: <http://www.cse.msu.edu/~cbowen/>