Jake Taggart
Week 1 Day 2 Research HW

Day 2 Research

1. What are the 4 main pillars of Object Oriented Programing and give description of each and how they are applied.

   A: The four main pillars of OOP are inheritance, encapsulation, polymorphism, and abstraction.

   Inheritance describes classes/objects can pass on methods to children classes which generally are more specific than the parent classes. Parent classes/objects may have grandchildren classes and even more specific classes. However, multiple inheritance is not allowed - which is to say that a child class/object may not inherit qualities from multiple parent classes. The way inheritance is put into action is by using the keywords "extends" or "implements" depending on whether the parent is an interface or class.

   Encapsulation is the idea of hiding an object's data from the possibility of manipulation within its origin class. The idea is to prevent any unwanted party to be able to see the value(s) of an object's data. The biggest ways to implement encapsulation is by using access modifiers including the keywords "public", "private", "protected", and "private". "Public" means what it means - that an object's data is not hidden or restricted. "Private" means that the data of an object can only be accessed or viewed directly by the class that declares the object. "Protected" is a bit in between - meaning that certain parties/classes can access the data of an object based on the user's specification. "Default" automatically acts as "public". Using public getter and setter methods are ways to access an object's data without being able to directly manipulate or see its value(s), but still be able to set the object's data and get an object's data without interfering with an encapsulated object directly.

   Polymorphism is similar to inheritance in that it involves parent/child classes, but is different in that the child classes may override or overload the parent class's methods when necessary via the keywords "override" and "overload". Override allows the programmer to allow child classes to provide more specificity in the methods declared in the parent class. For example, if an animal class has a speak() method then its children classes may be a variety of animals such as "cat", "dog", "mouse", etc. Overriding ensures via the keyword "override" that based on which object is identified, the correct subclass/child class is chosen for the method instead of the parent's class method or its sibling classes. Overloading is when a method requires a different argument, meaning that by redefining a method's signature (arguments) allows the programmer to call a correct method based on the arguments used by the method.

Abstraction is the practice of hiding unnecessary information from the user whilst allowing said user to build upon the original object. For example, a car object can have complex functions such as drive(), etc. without the need for knowing exactly how the ins and outs of cars work. The way to use abstraction is via abstract classes and use of interfaces.

2. What are SOLID programming principles and what does each section detail?

A: SOLID programming principles include single responsibility as a principle, the open-closed principle, the Liskov substitution principle, interface segregation principle, and the dependency inversion principle. The single responsibility principle indicates that each class should only have one and only one task to complete. Each class should have one single purpose. For example, if one wants to calculate the area of various shapes then he/she needs to create a class for each since a circle would not be the same as a square and so on. The open-closed principle enforces encapsulation in a way by encouraging the programmer to make classes open to extension, but not directly modifiable. The Liskov substitution encourages the programmer to have subclasses inherit useful features from parent classes and building onto those features to adapt to the subclass's purpose. The interface-segregation principle simply encourages programmer's to make every interface/child class a uniquely purposeful element that serves some purpose. This is saying that a program should never have an implementation of an interface/subclass with no realistic use. The dependency inversion principle encourages the programmer to force entities to rely upon abstractions and not concretions. Thus, each class/object which is made should be made to serve a purpose but also to allow the further extension of purposes in subclasses, etc. Rather than reinventing the wheel, each entity should be focused on modularity and the idea of "less is more" programming.

3. What are the differences of the Following:

A: HashMaps vs. HashTables - The difference between the two is that HashMaps are nullable while HashTables are non-nullable and threadsafe. Thus, HashTables allow for concurrent manipulation of the object's data which is to say that multiple entities can manipulate the HashTable at once. HashMaps are non-threadsafe which means that only one entity can manipulate the HashMap at one time.

List vs. ArrayList - List is a more abstract collection compared to ArrayList. This means that List is functions like an interface while ArrayList functions as the implementation of that interface.

Array vs. ArrayList - An array is more restrictive in terms of the dynamics in memory. Thus an array has a static container for memory space meaning that the amount of memory spaces necessary is defined when creating an array and cannot change. An ArrayList, however, dynamically changes the memory spaces allowed. Thus, one can begin an ArrayList and continue adding values without having to worry about the overuse of memory spaces.

HashSet vs. HashMap - A HashMap can have duplicate values in its collection while a HashSet cannot have duplicate values. A HashMap also allows for the use of keys to indicate certain elements while HashSets' elements are not identified via keys.

StringBuilder vs StringBuffer - The main difference between StringBuilder and StringBuffer is that StringBuilder is not threadsafe while StringBuffer is threadsafe. Because of this, StringBuilder is quicker.

4. Why is it important to override the equals and hashCode methods for Java objects?

   A: The reason it's important to override the equals and hashCode methods for Java objects is because a hashCode always regenerates a new code for the denoted element. Thus, if equal() is not overridden while hashCode is then the hashCodes will not be truly accurate. In the reverse scenario, the same thing would happen.

5. What is the difference in an Abstract Class and an Interface?

   The main difference between an abstract class and an interface is that an interface's variables are final while and abstract class allows for non-final variables. This also means that an interface cannot have its methods implemented. Java abstract classes allow for methods to be instantiated which implements the default method used.