Md Rahman

Week1Day2 Research

1) What are the four main pillars of OOP and give description of each.

1. Inheritance. Done using extends and implements. Allows for reusable code. A child class (subclass) extends/inherits from a parent (superclass). The inherited class can use all the functions defined in the parent.
2. Encapsulation. A way of hiding and restricting access to important attributes such as class variables. Done using access modifiers such as public, private, protected and default. These attributes can be accessed and modified using getters and setters.
3. Abstraction - Showing only the essential features that interests the user in a simple manner, and hiding the complex stuff in the background. The user isn’t interested in all the behind-the-scenes actions; they just want an easy way of understanding the program. Done using Abstract classes and interfaces.
4. Polymorphism - Inherited classes may redefine methods that they extend from their parent class and even add their own unique attributes. However, this does not work the other way around - a parent class cannot have the exact same behavior as it’s subclass. Done with overloading and overriding.

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

1. Single responsibility principle. A class should only have one major job. For example, if a certain class will only be used for a specific calculation, it shouldn’t deal with outputting any data, as the format of the output may need to be different depending on the situation.
2. Open Closed Principle. Objects should be open for extension but closed for modification. A class should be easily extendable but should not be modifiable.
3. Liskov substitution principle. A way of ensuring that inheritance is used correctly so that whenever a new subclass is created, it can properly extend the attributes of its superclass.
4. Interface segregation principle. Clients should not be forced to implement an interface that they won’t use. It’s better to have smaller, simple to understand interfaces than larger, more complex ones.
5. Dependency Inversion Principle. Depend on abstractions, not on concretions. High level modules shouldn’t rely on lower level modules. Instead, both should rely on abstractions. Also, abstractions should not depend upon details. Details should depend upon abstractions.

3. What are the differences between the following:

1. HashMap vs HashTable.
   1. Hashtable is synchronized, making it better for threaded apps. Does not allow null keys or values.
   2. Hashmap is unsynchronized, making it not thread safe. It’s better for non-threaded apps and allows for one null key and any number of null values.
2. ArrayList vs List
   1. List acts as an interface whereas ArrayList is an implementation of that interface. The List interface includes some methods that are not found in ArrayList class. List creates a collection of elements which are stored in sequence and can be accessed by it’s index number, whereas ArrayList creates an array of objects where the size can grow and reduce dynamically.
3. Array vs ArrayList
   1. Array is used to hold multiple values of the same type. Usually used to hold primitive values of the same type. Can come in multiple dimensions. Must declare size beforehand; size does not dynamically grow.
   2. ArrayList can only contain objects, not primitives. ArrayLists can also grow dynamically in size. Also has an iterator to iterate thru the list.
4. HashSet vs HashMap
   1. HashSet is an implementation of Set interface which doesn’t allow for duplicate values. Objects stored in HashSet must override equals() to check for equality. Can have a single null value
   2. HashMap is an implementation of Map interface, which maps a key to a value. Can have single null key and any number of null values.
5. StringBuilder vs StringBuffer. These are mutable, whereas String is immutable.
   1. StringBuffer. An older way for String manipulation. All of its public methods are synchronized, meaning they’re thread safe.
   2. StringBuilder. Another way for String manipulation. Unsynchronized, making it non -thread safe but faster than StringBuffer.

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

I. We need to override so our class can properly function with all hash-based collections, including HashMap,HashSet and HashTable. If two objects are equal, their hashcodes must also be equal. The default hashcode method in class Object always returns a unique number for each object, so even if you do override equals(), it doesn’t make a difference unless their hashcodes also reflect this equivalency.

5. Difference between an Abstract Class and Interface.

1. Abstract classes can have both abstract and non abstract methods. Implemented using extends keyword. Subclasses extending an abstract class may inherit some or all of it’s methods.
2. Interface can only have abstract methods. Implemented using “implements” keyword. Subclass implementing an interface MUST use all methods in the interface.