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

The four main pillars of object oriented programming is Abstraction, Encapsulation, Inheritance, Polymorphism.

Abstraction

Let’s start with abstraction. Abstraction is the process of hiding information from the user in a manner that only relevant information is displayed. In Java this is mostly done through using abstract classes and interfaces. Abstraction is necessary as it helps reduce the complexity of a problem.

Encapsulation

For encapsulation, it can best be summarized as limiting what information is available and how someone can access it.

This can be done through the four access modifiers, Public, Private, Protected and Default.

Public has the widest scope out of the four modifiers, which means that this can be accessed anywhere in the program. This is mostly used at the class level, and for specific methods that provide an entry point to that classes member variables. An example of these methods are getters and setters.

Private however, is the exact opposite. Private limits the scope only to the same class. This limits the visibility only to the class. Even subclasses cannot see information that is declared private.

Protected is similar to private, but allows a subclass to be able to see and manipulate the data of its superclass.

A default access modifier means that the information is visible within the same package

Inheritance

Inheritance is the passing of fields and methods from a superclass to a subclass.

In java, multiple inheritance is not supported, meaning that a class cannot have more than one super class.

Polymorphism

Means that something can take many forms. In Java, this can be done through method overloading and method overriding.

Method overloading is using the same method name but with using different parameters.

Method overriding is having the same method signature between a superclass and a subclass, but having a different definition between the two. This can be done using an annotation above the method.

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

SOLID programming principles stands for can be broken up into five ideas

S Single-responsibility principle

Every class in a program should have only one reason to change.

O Open-Closed principle

Open for extension, closed for modification. Data members should be private and should only be accessed by getters and setters only when necessary.

L Liskov substitution principle

Objects should be replaceable by their subtypes without altering the correctness of the program

I Interface segregation principle

Using multiple interfaces that’s are specific to individual clients. Clients should not need to implement methods that would never be used by them.

D Dependency Inversion Principle

Everything should rely on abstraction

3. What are the differences of the Following:

HashMap vs HashTable

A HashMap allows null values to be included in a key while a Hashtable does not.

A HashMap is asychronized(not thread safe), while a HashTable is synchronized(Thread safe)

ArrayList vs List

A List cannot be instantiated. It can however, be defined as an ArrayList

A List is an interface

An ArrayList is an actual class that also implements a List.

An ArrayList can use its own methods as well as use the methods it inherited through list.

Array vs ArrayList

An ArrayList is dynamic, the size of it can change

An Array however, can only be initialized to one size.

HashSet vs HashMap

A HashMap can have duplicate values but it cannot have duplicate values in its key.

A HashSet can have only one null value, whereas a hashMap can have any number of null values, but only one null value in its key.

StringBuilder vs StringBuffer

String buffer is synchronous while String Builder is not

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

It’s important to override the equals and hashCode due to how objects can be stored using their hashcode. For instance if you use a hashMap or hashTable. If you do not overwrite hashcode anywhere where you override .equals, it prevents your class from functioning on the above.

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

Though functionally similar, there are differences between an Abstract Class and an Interface which make them useful in different situations. One major key difference between an abstract class and an interface is that an abstract class can have non-static, non final variables, whereas in an interface all variables are final and can only be static. Its easier to think of an interface as a contract, where the class that uses the interface must have a definition for each of the methods.