Research

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

The 4 main pillars of OOP are:

1. Polymorphism: An object’s ability to take on many forms. If a Java object passes the IS-A test, it is considered to be polymorphic. An example of this would be a Car and Vehicle class. Since a Car IS-A Vehicle, this is called polymorphism. Here is an example:

public class Vehicle{}

public class Car extends Vehicle{}

A Car IS-A Vehicle

A Car IS-A Object

The example above shows that the Car class is considered to be polymorphic since it passes more than 1 IS-A test.

1. Encapsulation: A process of data hiding by wrapping up data and method into a single class. Its goal is to provide data security by making it safe from outside access and misuse. Information hiding is achieved through public, private and protected access control modifiers. In other words, class variables will be hidden from other classes and are only accessible through methods of its current class, called getters and setters. Let’s say we have a class called Bank, which contains sensitive account information of a Customer. In the Customer’s class, we want to make the class variables private. To access each customer’s data, we will have to use getters and setters in the Customer class.
2. Inheritance: A process where one class inherits/receives another class’ properties. A subclass can only inherit from one super/parent class. To apply inheritance, the extends keyword is used on a subclass. An example of this would be an Employee and Person class.

public class Person{}

public class Employee extends Person{}

All the properties of Person are available for Employee to use. Super and this keywords are usually used to differentiate the members of a superclass from subclass.

1. Abstraction: is a process in which implementations are hidden from the user, but the functionality is provided to the user. Abstraction is done through Abstract classes and Interfaces. An example of this would be when a person drives a car. He or she may not know how the engine works or what happens when she hits the brake. A regular person would only know how to drive a car, not the complexity of how a car engine works. Another example would be:

Interface Animal{

public void run();

}

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

The SOLID principles are the following:

1. Single Responsibility: a class should only have one responsibility and should only have 1 reason to change. The benefits of this are to help with testing, lower coupling and organization. It will have fewer test cases, less functionality in a single class will have fewer dependencies, and are easier to search in smaller, well organized classes.
2. Open/Closed: Open for Extension, Closed for Modification. In a working application, one should stop modifying existing code and causing potential bugs. However, there needs to be modification when fixing bugs in an existing code.
3. Liskov Substitution: is a process in which if class A is a subtype of class B, then we should be able to replace B with A without disrupting our program behavior.
4. Interface Segregation: Larger interfaces should be divided into smaller ones. The benefit of this is to ensure that classes are implemented according to the methods that are of interest to them.
5. Dependency Inversion: this principle refers to the decoupling of software modules. An example of this would be a Computer tightly coupled with a StandardKeyboard and Monitor classes. Since there are many other keyboards out there and different type of monitors, we lost the ability to switch out any of these classes from the Computer class. Instead of this, the right way to do is to create a Keyboard interface that is injected into the Computer class and so a StandardKeyboard class can implements the Keyboard interface.

3. What are the differences of the Following:

**HashMap vs HashTable**

HashMap is asynchronized (not thread-safe), it allows one null key and multiple null values, faster, traversed by Iterator, and it inherits AbstractMap class.

HashTable is synchronized (thread-safe) and can be shared by many threads, HashTable does not allow any null key or value, slower, traversable by Iterator and Enumerator, inherits Dictionary class.

**ArrayList vs List**

List is:

1. An interface which extends Collection
2. It is used to create a list of elements or objects.
3. It can be accessed or associated by their index numbers
4. List does not have contains method.
5. The namespace for List interface is System.Collection.Generic

ArrayList is:

1. A class that extends AbstractList class and implements the List interface.
2. It is used to create dynamic arrays that can grow and shrink when needed.
3. The namespace is System.Collection.
4. It has contains method

**Array vs ArrayList**

Array is:

1. A fixed length data structure
2. Can contain both primitives and objects
3. Not resizable.
4. Can be iterated using for or foreach loop.
5. Length() returns the number of elements in the array.
6. Can be multi-dimensional

ArrayList is:

1. A variable length collection class.
2. Cannot contain primitives.
3. Dynamic in size.
4. Can use iterator to iterate through ArrayList.
5. Size() method returns the size or number of elements of the ArrayList.
6. Always single dimensional.

**HashSet vs HashMap**

HashSet:

1. Implements the Set interface
2. Does not allow duplicate elements
3. Allows a single null value

HashMap:

1. Implements the Map interface
2. Does not allow duplicate keys, but allows duplicate values.
3. Allows a single null key and any number of null values.

**StringBuilder vs StringBuffer**

StringBuilder:

1. Is asynchronized and not thread-safe
2. More efficient
3. Faster

StringBuffer:

1. Is synchronized and thread-safe
2. Less efficient
3. Slower

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

It is important to override the equals and hashCode methods for Java objects because if we don’t override them, two objects are considered equal ONLY if their references point to the same object. Furthermore, it is also important to always override hashCode() method if the class overrides equals(). Overriding only equals() method without overriding hashCode() causes the two equal instances to have unequal hash codes.

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

Interface:

1. Methods in an interface are implicitly abstract.
2. Variables in an interface are by default final.
3. Members are public by default.
4. “implements” is the keyword to implement an interface.
5. Can only extend another Java interface only.

Abstract Class:

1. Can have instance methods that implements a default behavior.
2. May contain non-final variables
3. Can have private, protected and public members.
4. “extends” is the keyword used.
5. Can extend another Java class and implement multiple Java interfaces.