# HW#1

**1 –** In order to take advantage of OOP concepts as better organized, maintainable, reusable, modular code. C++, Java, C#, Python, R, PHP, JavaScript etc.

**2-** An abstract class allows us to create functionality that subclasses can implement or override. An interface only allows us to define functionality, not implement it. And whereas a class can extend only one abstract class, it can take advantage of multiple interfaces.

**3-** The equals() and hashcode() are both important methods provided by the **Object** class for **comparing** objects.
Object.hashCode() needs to be overriden in cases Object.equals() is overriden.
The reason for this is that hash based collections relies on Object.hashCode() which will be inaccurate for cases that Object.equals() is modified.
Override is to be executed when the inherited method does not function as we want it to or needs modification for the sub-class.

**4 –** Java doesnt support multi-inheritance by default as the diamond problem is simply not being able to inherit multiple parent classes.
the solution to the diamond problem is default methods and interfaces. we can achieve multiple inheritance by using these two things.

**5-** The garbage collector provides the following benefits: Frees developers from having to manually release memory. Allocates objects on the managed heap efficiently. Reclaims objects that are no longer being used, clears their memory, and keeps the memory available for future allocations.

**6-** When we want to have variables that are common to all objects. This is accomplished with the static modifier. Fields that have the static modifier in their declaration are called *static fields* or *class variables*. They are associated with the class, rather than with any object. Every instance of the class shares a class variable, which is in one fixed location in memory. Any object can change the value of a class variable, but class variables can also be manipulated without creating an instance of the class.

**7 -** The immutable objects are **objects whose value can not be changed after initialization**. We can not change anything once the object is created. This can be achieved by making all fields as final, by not allowing subclasses to override methods. The simplest way to do this is to declare the class as final. A more sophisticated approach is to make the constructor private and construct instances in factory methods, by not providing "setter" methods — methods that modify fields or objects referred to by fields.

**8 -** Aggregation
It is a special form of Association where:
It represents Has-A's relationship.
It is a unidirectional association i.e. a one-way relationship. For example, a department can have students but vice versa is not possible and thus unidirectional in nature.

In Aggregation, both the entries can survive individually which means ending one entity will not affect the other entity.

______________

Composition is a restricted form of Aggregation in which two entities are highly dependent on each other.
It represents part-of relationship.
In composition, both entities are dependent on each other.
When there is a composition between two entities, the composed object cannot exist without the other entity.

**9 -** Cohesion refers to what the class (or module) can do. Low cohesion would mean that the class does a great variety of actions - it is broad, unfocused on what it should do. High cohesion means that the class is focused on what it should be doing, i.e. only methods relating to the intention of the class. As for coupling, it refers to how related or dependent two classes/modules are toward each other. For low coupled classes, changing something major in one class should not affect the other. High coupling would make it difficult to change and maintain your code; since classes are closely knit together, making a change could require an entire system revamp.

Good software design has high cohesion and low coupling.

**10 -** The stack is the memory set aside as scratch space for a thread of execution. When a function is called, a block is reserved on the top of the stack for local variables and some bookkeeping data. When that function returns, the block becomes unused and can be used the next time a function is called. The stack is always reserved in a LIFO (last in first out) order; the most recently reserved block is always the next block to be freed. This makes it really simple to keep track of the stack; freeing a block from the stack is nothing more than adjusting one pointer.

The heap is memory set aside for dynamic allocation. Unlike the stack, there's no enforced pattern to the allocation and deallocation of blocks from the heap; you can allocate a block at any time and free it at any time. This makes it much more complex to keep track of which parts of the heap are allocated or free at any given time; there are many custom heap allocators available to tune heap performance for different usage patterns.

**11-** When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.
The first kind of exception is the *checked exception*. These are exceptional conditions that a well-written application should anticipate and recover from.
The second kind of exception is the *error*. These are exceptional conditions that are external to the application, and that the application usually cannot anticipate or recover from.
The third kind of exception is the *runtime exception*. These are exceptional conditions that are internal to the application, and that the application usually cannot anticipate or recover from. These usually indicate programming bugs, such as logic errors or improper use of an API.

**12 -** Clean code is code that is easy to understand and easy to change.

**13 –** Method hiding may happen in any hierarchy structure in java. When a child class defines a static method with the same signature as a static method in the parent class, then the child's method *hides* the one in the parent class.
Hiding doesn't work like overriding, because static methods are not polymorphic. Overriding occurs only with instance methods. It supports late binding, so which method will be called is determined at runtime.

On the other hand, method hiding works with static ones. Therefore it's determined at compile time.

**14 -**
- Abstraction allows a programmer to design software better by thinking in general terms rather than specific terms while Polymorphism allows a programmer to defer choosing the code you want to execute at runtime.

- Abstraction is implemented using abstract class and interface in Java while Polymorphism is supported by overloading and overriding in Java.

- Though overloading is also known as compile-time Polymorphism, method overriding is the real one because it allows a code to behave differently at different runtime conditions, which is known as exhibiting polymorphic behavior.