**Ans Q1(a)**

**Tight Coupling –**

* The **tightly coupled object** is an object that needs to know about other objects and is usually highly dependent on each other's interfaces.
* Changing one object in a **tightly coupled application** often requires changes to a number of other objects.
* In the small applications, we can easily identify the changes and there is less chance to miss anything. But in large applications, these inter-dependencies are not always known by every programmer and there is a chance of overlooking changes.

**Loose Coupling –**

* **Loose coupling** is a design goal to reduce the inter-dependencies between components of a system with the goal of reducing the risk that changes in one component will require changes in any other component.
* **Loose coupling** is a much more generic concept intended to increase the flexibility of the system, make it more maintainable and makes the entire framework more stable.

**Ans Q1(b) –**

**High Cohesion –**

* High Cohesion is when you have a class that does well-defined job.
* High Cohesion gives us better-maintaining facility.
* Example -

Class A

checkEmail();

Class B

validateEmail();

Class C

sendEmail();

Class D

printLetter();

**Low Cohesion –**

* Low Cohesion is when a class does a lot of jobs that don’t have much in common.
* Low Cohesion result in monolithic class that are difficult to maintain, understand and reduce re-usability.
* Example –

Class A

checkEmail();

validateEmail();

sendEmail();

printLetter();

printAdress();

**Ans 2(a) –**

**DRY Principle –**

**DRY stand for “Don’t Repeat Yourself”, a basic principle of software development aimed at reducing repetition of information. The DRY principle is stated as, “Every piece of knowledge or logic must have a single, representation within a system”.**

**We can achieve DRY by divide your system into piece. Divide your code and logic into smaller reusable units and use that code by calling it where you want. Don’t write lengthy methods, but divide logic and try to use the existing piece in your method.**

**Ans 2(b) –**

**KISS Principle –**

**KISS stand for “Keep It Simple, Stupid”, Kiss principle helps in keeping code simple and clear, making it easy to understand. As we know programming language are for humans to understand – computers can only understand 0 and 1 – so keep coding and methods simple and straightforward. Each method** **should never be more than 40-50 lines.**

**Each method should only solve one small problem, not many use cases. If you have a lot of conditions in the method, break these out into smaller methods. It will not only be easier to read and maintain, but it can help find bugs a lot faster.**

We can achieve this by trying to write simple code. Think of many solutions for your problem, then choose the best, simplest one and transform that into your code. Whenever you find lengthy code, divide that into multiple methods — right-click and refactor in the editor. Try to write small blocks of code that do a single task.

**Ans 3.**

**Pessimistic Locking –**

Pessimistic Locking is a way to of achieving mutual exclusion by always locking the entire scope.The first thread to acquire the lock will retain the lock until the scope execution. It will then release the lock which can then be acquired by any other waiting thread.

The problem with this approach lies in the fact that we always assume that all possible threads will be competing for the underlying resource at the same time which is generally not the case.

Due to this assumption, the additional overhead of acquiring and releasing the lock comes into the picture even if there is a single thread that is repeatedly executing the synchronized block.

**Optimistic Locking –**

Optimistic locking is a way of managing concurrency control, generally used by transactional systems. In this implementation, we allow multiple threads to compete for the update completion but only committing the transaction if it's not already updated.

Ans 4.

**Deadlock –**

A deadlock is a situation in which two computers [programs](https://www.techtarget.com/searchsoftwarequality/definition/program) sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function.

In older times some operating systems offered dynamic allocation of resources. Programs could request further allocations of resources after they had begun running. This led to the problem of the deadlock.

However, now neither program can proceed until the other program releases a resource. The operating system cannot know what action to take. At this point the only alternative is to abort (stop) one of the programs.

Few methods we can use to handle deadLock are –

1. DeadLock ignorance
2. DeadLock prevention
3. DeadLock avoidance

Ans 5.

HTTP stands for “Hypertext Transfer Protocol”, It is used for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes.

It follows a classical [client-server model](https://en.wikipedia.org/wiki/Client%E2%80%93server_model), with a client opening a connection to make a request, then waiting until it receives a response. HTTP is a [stateless protocol](https://en.wikipedia.org/wiki/Stateless_protocol), meaning that the server does not keep any data (state) between two requests.

Difference between Http (HypertextTransfer Protocol) and Https (HypertextTransfer Protocol Secure’) –

* HTTP lacks security mechanism to encrypt the data whereas HTTPS provides SSL or TLS Digital Certificate to secure the communication between server and client.
* HTTP operates at Application Layer whereas HTTPS operates at Transport Layer.
* HTTP by default operates on port 80 whereas HTTPS by default operates on port 443.
* HTTP transfers data in plain text while HTTPS transfers data in encrypt text.
* HTTP is fast as compared to HTTPS because HTTPS consumes computation power to encrypt the communication channel.