Q3. What is the Singleton Design Pattern? Explain with an example and where can it be used? Also Explain eager and Lazy initialization.

Singleton Design Pattern : - Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

Eager initialization = This is the simplest method of creating a singleton class. In this, object of class is created when it is loaded to the memory by JVM. It is done by assigning the reference of an instance directly.   
It can be used when program will always use instance of this class, or the cost of creating the instance is not too large in terms of resources and time.

Lazy initialization = In this method, object is created only if it is needed. This may prevent resource wastage. An implementation of getInstance() method is required which return the instance. There is a null check that if object is not created then create, otherwise return previously created. To make sure that class cannot be instantiated in any other way, constructor is made final. As object is created with in a method, it ensures that object will not be created until and unless it is required. Instance is kept private so that no one can access it directly.   
It can be used in a single threaded environment because multiple threads can break singleton property as they can access get instance method simultaneously and create multiple objects.

Q1 We want to design a social media application which allows me to connect to my friends on all four social platforms i.e. Facebook, Google Plus, Twitter and Orkut (for example sake). Now I want that client to be able to tell the name of friend and desired platform – then my application should connect to him transparently. More importantly, if I want to add more social platforms into an application then application code should accommodate it without breaking the design.

function SocialMediaDesignPattern() {

    this.sitename = ""

    this.setSocialmediaStrategy = (site) => {

        this.sitename = site.socailsite

    }

    this.join = (username) => {

        console.log(`Hey ${username} welcome to ${this.sitename}`)

    }

}

function GooglePlusStrategy() {

    this.socailsite = "Google Plus"

}

function TwitterStrategy() {

    this.socailsite = "Twitter"

}

function FacebookStrategy() {

    this.socailsite = "Facebook"

}

function OrkutStrategy() {

    this.socailsite = "Orkut [not possible though :)]"

}

let context = new SocialMediaDesignPattern();

// Setting Facebook strategy

context.setSocialmediaStrategy(new FacebookStrategy());

context.join("Ankush Gupta");

// Setting Twitter strategy

context.setSocialmediaStrategy(new TwitterStrategy());

context.join("Ankush Gupta");

// Setting GooglePlus strategy

context.setSocialmediaStrategy(new GooglePlusStrategy());

context.join("Ankush Gupta");

// Setting Orkut strategy

context.setSocialmediaStrategy(new OrkutStrategy());

context.join("Ankush")