1. **What is microservice? Briefly explain microservice with examples.**

   Microservice is an architectural style that develops a single application as a set of services. Each service runs in its own process. The services communicate with clients, and often each other, using lightweight protocols, often over messaging or HTTP.

   Suppose we are building an application which will collect user daily activity data and based on the data it will suggest food for the user. Here we can think of two services.

   1. User data collection service
   2. AI Model train & provide food suggestion service

   Both services are independent of each other and can use suitable technology to develop.
   So, the advantages of microservices are: Improve productivity, Improve scalability, can use different technology for different services etc.

2. **What is dispatcher-servlet? Why / when is it used? Shortly explain how spring-boot lifecycle works.**

   Dispatcher-servlet also known as Front Controller design pattern, a single controller is responsible for directing incoming HttpRequests to all of an application's other controllers and handlers.

   Spring's DispatcherServlet implements this pattern and is, therefore, responsible for correctly coordinating the HttpRequests to their right handlers.

3. **What is git merge and what is git rebase? What are the differences between those? When will you choose git merge and when will you choose git rebase?**

   Git Merge
   Merging takes the contents of a source branch and integrates them with a target branch. In this process, only the target branch is changed. The source branch history remains the same.

   Git Rebase
   Rebase is another way to integrate changes from one branch to another. Rebase compresses all the changes into a single “patch.” Then it integrates the patch onto the target branch.

   Unlike merging, rebasing flattens the history because it transfers the completed work from one branch to another. In the process, unwanted history is eliminated.

   When we need to keep track of each commit's merging history, we will use git merge. When you don't need to keep merging history, recommended to use rebase.

4. **Suppose you made a wrong commit and pushed the code into your remote branch. Now you feel the push needs to be reverted or the previous head should be back into the remote branch. What would you like to do?**

   If you have a review server then you can abandon the commit from the review server.
   On the other hand you can check out to the current commit and undo the changes and push again to the remote server.

5. **Suppose you started a pizza restaurant with two main types, margarita and classic pizzas. Once customers start coming in, they demand add-ons like mushrooms, onions etc. To save the day, you created subclasses for mushroom and onion. But shortly after, a competitor opens a new restaurant nearby with subclasses for corn, olives, etc. Now considering new competitors, a number of subclasses you will need to create for an effective billing system can go overboard. Which design pattern do you think will be fit most to solve this problem any why? Please briefly explain.**

   This will be so much easier and smaller if it uses the decorator pattern.

   The decorator design pattern falls into the structural category, that deals with the actual structure of a class, whether by inheritance, composition or both. The goal of this design is to modify an objects' functionality at runtime. This is one of the many other design patterns that utilize abstract classes and interfaces with composition to get its desired result.

   In this scenario we need to change the object behavior in the runtime. Decorator design pattern will allow us to do that easily.