**1. JavaBank Case Study Installation and Exploration**

To install the JavaBank Case Study and explore its functionality, follow these steps:

1. **Installation:**
   * If you have not installed JavaBank, refer to the slides from Slide 6 of the provided lesson materials.
   * Follow the steps to download, install, and set up the JavaBank application on your system.
2. **Exploring JavaBank:**
   * **Display Accounts:**
     + If no accounts are created, attempting to display accounts typically shows an empty list or a message indicating no accounts exist.
     + If accounts are created, the system should list the details of each account (e.g., account number, holder name, balance).
   * **Create Accounts:**
     + Entering valid details in the required fields creates a new account.
     + If you leave fields empty, the application may prompt you to fill in the required fields or prevent account creation.
   * **Delete Accounts:**
     + Deleting an account removes it from the list of accounts. The account's data is typically no longer retrievable after deletion.
   * **Make a Withdrawal Transaction:**
     + If the withdrawal amount is specified and sufficient funds are available, the transaction should decrease the account balance.
     + If no amount is specified, the system may prompt for an amount or prevent the transaction.
   * **Make a Deposit Transaction:**
     + Depositing funds into an account increases the balance by the specified amount.
     + Similar to withdrawals, the system may require an amount to be specified before processing the deposit.
   * **Edge Cases and Observations:**
     + Displaying accounts before any are created generally results in an empty list.
     + Attempting to create an account without entering data in required fields may cause the application to throw an error or prompt the user.
     + Withdrawal or deposit transactions with no amount specified may not be processed, with the system prompting for valid input.
     + Additional questions might include:
       - How does the system handle large numbers of accounts?
       - What are the performance implications of frequent transactions?
       - Are there security measures for protecting account information?
   * **Suggestions for Improvement:**
     + **Error Handling:** Improve error messages to be more user-friendly and informative.
     + **Validation:** Add more robust validation for account creation and transactions to prevent invalid inputs.
     + **User Interface:** Consider enhancing the UI for better usability, possibly including a dashboard view.
     + **Functionality:** Adding features such as transaction history, account search, and export options could increase the application's utility.

**3. Bike Project Analysis**

If you import the bikeproject.zip file and explore the code:

* **a. Primitive Data Type Example:**
  + int numGears might be an example of a primitive data type used within a bike class to store the number of gears.
* **b. String Concatenation Example:**
  + Concatenation may occur in a method like toString() where the class returns a string representation of the object, e.g., return "Bike with " + numGears + " gears";.
* **c. Object Names:**
  + Objects could be named like mountainBike, roadBike, etc., depending on the bike types instantiated in the program.
* **d. Constructors in Each Class:**
  + Each bike class might have multiple constructors: a default constructor and one or more parameterized constructors.
* **e. Inheritance Structure:**
  + **Superclass:** Bike could be the superclass.
  + **Subclasses:** MountainBike and RoadBike might extend Bike, indicating inheritance.
* **f. Standard Bike Sample Values:**
  + **Mountain Bike (MB):**
    - handleBars: Bull Horn
    - suspension: RockShox XC32
    - tyreWidth: 20
  + **Road Bike (RB):**
    - handleBars: Drop
    - Other values based on defaults defined in the program.

**4. Calculator Program Modifications**

**Steps:**

* **a. Import and Run:**
  + Follow the instructions to import the Calculator.jar file into Eclipse.
  + Run the application using CalcMain.
* **b. Investigation:**
  + The Calculator should perform basic arithmetic operations like addition.
* **c. Add Multiplication and Subtraction:**
  + Modify the UI to add buttons for multiplication (\*) and subtraction (-).
  + Implement the logic to handle these operations by updating the code handling button clicks.
* **d. Test Functionality:**
  + Ensure all operations (addition, subtraction, multiplication) work correctly.
* **e. Export Runnable JAR:**
  + From Eclipse, export the modified project as a runnable JAR file.
  + Double-click the JAR file to ensure the application runs independently with all features functioning as expected.