SOFTWARE CONSTRUCTION AND DEVELOPMENT- LAB

By
Samia Aziz
Instructor SE
Samia.aziz@nu.edu.pk

Lab 8- Git & Github

- **You will complete this lab in a group of 2-3 members max.**
- **Only one member needs to submit the lab.**

Task 1: Fast Forward Merging

Addition, Multiplication, and Input Validation Feature

Scenario:
You and your teammate are working on a Calculator class. One member will implement both the addition() and multiplication() methods, while the other member will handle input validation and merging all the changes into the main branch. Both members will work on separate branches, and the merging process should use fast forward.

Group Roles:

- **Member 1:** Add both addition() and multiplication() methods.
- **Member 2:** Add input validation for both methods and handle fast forward merging.

Steps:

Member 1:

1. **Create a new branch:**
   - In Eclipse, create a branch called feature/calculator-methods and check it out.
2. **Add the addition feature:**
   - Implement the addition() method in the Calculator class, which takes two integer inputs and returns their sum.
3. **Add the multiplication feature:**
   - Implement the multiplication() method in the Calculator class, which takes two integer inputs and returns their product.
4. **Commit the changes:**
   - Commit your changes with the message: "Added addition and multiplication methods."
5. **Push the branch to GitHub:**
   - Push the feature/calculator-methods branch to the remote repository on GitHub.

Member 2:

1. **Create a new branch:**
   - Create and check out a branch named feature/validation.
2. **Add input validation:**
   - Implement input validation in the Calculator class to ensure that inputs for both addition() and multiplication() are integers and non-negative.
   - Modify the addition() and multiplication() methods to incorporate this validation.
3. **Commit the changes:**
   - Commit your changes with the message: "Added input validation for addition and multiplication methods."
4. **Push the branch to GitHub:**
   - Push the feature/validation branch to the remote repository.
5. **Switch to the main branch:**
   - After pushing, switch to the main branch in Eclipse.
   - Pull the latest changes from the main branch to ensure it's up-to-date.
6. **Merge the calculator methods branch:**
   - Merge the feature/calculator-methods branch into the main branch using fast forward merge.
7. **Merge the validation branch:**
   - Next, merge the feature/validation branch into the main branch using fast forward merge.
8. **Push the merged changes:**
   - Push the updated main branch to GitHub after verifying that both the addition() and multiplication() methods, as well as the validation logic, work correctly.

Task 2: Manual Merging

Conflict in UI Design

Scenario:
You and your teammate are modifying the designUI() method in different ways on separate branches. You'll need to resolve the conflict manually while ensuring that both design aspects are incorporated into the final UI.

Group Roles:

- **Member 1:** Modify the UI design (e.g., button colors, font size).
- **Member 2:** Modify the UI design differently (e.g., layout changes).
- **Member 3:** Resolve the merge conflict and ensure the UI functions as intended.

Steps:

Member 1:

1. **Create a branch for your design:**
   - In Eclipse, create a branch called feature/design1 and check it out.
2. **Modify the UI design:**
   - Change the designUI() method to update the button colors to a new color scheme (e.g., change from blue to green).
   - Adjust the font size for buttons to improve visibility.
3. **Commit and push changes:**
   - Commit the changes with the message: "Updated button design and font size."
   - Push the feature/design1 branch to GitHub.

Member 2:

1. **Create another branch for your design:**
   - Create and check out a branch called feature/design2.
2. **Modify the UI design differently:**
   - Change the designUI() method to update the button colors to a new color scheme (e.g., change from green to red).
   - Add new components, such as a new label or input field, to enhance functionality.
3. **Commit and push changes:**
   - Commit your changes with the message: "Updated layout design and added new components."
   - Push the feature/design2 branch to GitHub.

Member 3 (Conflict Resolver):

1. **Switch to the main branch and pull changes:**

   - In Eclipse, switch to the main branch and pull the latest changes to ensure you have the most recent updates.
2. **Merge branch feature/design1:**
   - Merge feature/design1 into main. This should succeed without conflicts.
   - Verify that the button design is as intended.
3. **Merge branch feature/design2:**
   - Attempt to merge feature/design2 into main. A conflict will occur in the designUI() method.
4. **Resolve the conflict manually:**
   - Use Eclipse's conflict resolution tool:
     - Review the conflicting code sections.
     - Combine both the button color changes and layout modifications into a single cohesive designUI() method.
     - Ensure the new UI maintains both design aspects and test functionality.
5. **Commit and push:**
   - After resolving the conflict and verifying the UI, commit the merged changes with the message: "Resolved conflict in designUI() and merged both design changes."
   - Push the updated main branch to GitHub.

Task 3: Manual Merging

Conflict in Business Logic

Scenario:
You and your teammate are working on different improvements to the validateUser() method, which leads to a conflict that needs manual resolution while maintaining the integrity of the user validation and registration process.

Group Roles:

- **Member 1:** Add validation logic for user credentials.
- **Member 2:** Improve the registration logic within the same method.
- **Member 3:** Resolve the conflict and ensure the functionality is seamless.

Steps:

Member 1:

1. **Create a new branch:**
   - In Eclipse, create a branch called feature/validation and check it out.
2. **Add validation logic:**
   - Modify the validateUser() method to include new validation rules, such as checking if the username meets certain length requirements and if the password contains special characters.
   - Add error handling for invalid inputs.
3. **Commit and push:**
   - Commit the changes with the message: "Added validation logic for username and password."
   - Push the feature/validation branch to GitHub.

Member 2:

1. **Create another branch:**
   - Create and check out a branch called feature/registration.
2. **Improve registration logic:**
   - Modify the validateUser() method to enhance the registration flow, such as checking for duplicate usernames and handling email validation.
   - Add logging for user actions to track registration attempts.
3. **Commit and push:**
   - Commit the changes with the message: "Improved registration flow and added logging."
   - Push the feature/registration branch to GitHub.

Member 3 (Conflict Resolver):

1. **Switch to main and pull changes:**
   - In Eclipse, switch to the main branch and pull the latest changes to ensure it is up to date.
2. **Merge branch feature/validation:**
   - Merge feature/validation into main. This should succeed without conflicts.

3. **Merge branch feature/registration:**
    - Attempt to merge feature/registration into main. A conflict will occur in the validateUser() method.
4. **Resolve the conflict manually:**
    - Use Eclipse's conflict resolution tool:
        - Review the conflicting sections in the validateUser() method.
        - Combine both the new validation rules and improved registration logic to create a cohesive function that validates user inputs and handles registrations correctly.
        - Ensure all logic paths are covered and the functionality flows seamlessly.
5. **Commit and push:**
    - After resolving the conflict and testing the method for correctness, commit the merged changes with the message: "Resolved conflict in validateUser() and merged validation and registration logic."
    - Push the updated main branch to GitHub.

Task 4: Recursive Merging

Complex Merge Between Multiple Features

Scenario:
You and your teammates are collaboratively enhancing the BankAccount class by adding multiple new methods: withdraw(), deposit(), and accountBalance(). Each team member will work on a different method, and you'll need to handle a recursive merge that requires resolving conflicts due to interdependencies between the methods.

Group Roles:

- **Member 1:** Add the accountBalance() method.
- **Member 2:** Add the withdraw() method.
- **Member 3:** Add the deposit() method and handle the merge.

Steps:

Member 1:

1. **Create a branch for your feature:**

   - In Eclipse, create a branch called feature/balance and check it out.
2. **Add the accountBalance() method:**
   - Implement the accountBalance() method in BankAccount.java, which returns the current balance of the account.
   - Ensure that this method retrieves data from the class variable that stores the account balance.
3. **Commit and push changes:**
   - Commit your changes with the message: "Added account balance method."
   - Push the feature/balance branch to GitHub.

Member 2:

1. **Create a branch for your feature:**
   - Create a branch called feature/withdraw and check it out.
2. **Add the withdraw() method:**
   - Implement the withdraw() method in BankAccount.java, which deducts a specified amount from the account balance.
   - Ensure that the method checks for sufficient funds before processing the withdrawal.
3. **Commit and push changes:**
   - Commit your changes with the message: "Added withdraw method."
   - Push the feature/withdraw branch to GitHub.

Member 3:

1. **Create a branch for your feature:**
   - Create a branch called feature/deposit and check it out.
2. **Add the deposit() method:**
   - Implement the deposit() method in BankAccount.java, which adds a specified amount to the account balance.
   - Ensure this method validates the input to prevent negative deposits.
3. **Commit and push changes:**
   - Commit your changes with the message: "Added deposit method."

  - Push the feature/deposit branch to GitHub.

4. **Pull the latest changes from main:**
   - Switch to the main branch and pull the latest changes to ensure you have the most recent updates, including the newly added methods from your teammates.

5. **Merge the feature branches:**
   - **Merge feature/balance into main:**
     - Attempt to merge feature/balance into main. This should succeed without conflicts if the accountBalance() method does not conflict with existing code.
   - **Merge feature/withdraw into main:**
     - Attempt to merge feature/withdraw into main. If conflicts occur, resolve them:
       - Use Eclipse's conflict resolution tool to analyze and merge changes between withdraw() and accountBalance() if they access shared variables or methods.
       - Ensure that the withdraw() method integrates correctly with the existing functionality of accountBalance().
   - **Merge feature/deposit into main:**
     - Attempt to merge feature/deposit into main. Resolve any conflicts:
       - Analyze and resolve any conflicts that arise between deposit() and both withdraw() and accountBalance().
       - Ensure all methods function cohesively within the BankAccount class, considering how deposits may affect the balance shown by accountBalance().

6. **Test the integrated methods:**
   - After resolving all conflicts, run tests to verify that:
     - The withdraw() method deducts from the balance correctly.
     - The deposit() method adds to the balance correctly.
     - The accountBalance() method reflects the current state of the account correctly after deposits and withdrawals.

7. **Commit and push the merged changes:**
   - Once everything is working correctly, commit the merged changes with the message: "Merged all features and resolved conflicts in BankAccount class."
   - Push the updated main branch to GitHub.