Banking Application (Java, Maven, JUnit 5)
1. Project Overview Valeriia Holotiuk

This project is a Simple Banking Application built in Java. It demonstrates object-oriented programming, unit testing with JUnit 5, and continuous integration using GitHub Actions.

Features:
- Create accounts with unique IDs and initial balances
- Deposit and withdraw with validation
- Transfer money between accounts
- Balance inquiry
- Duplicate account prevention

2. How It Works

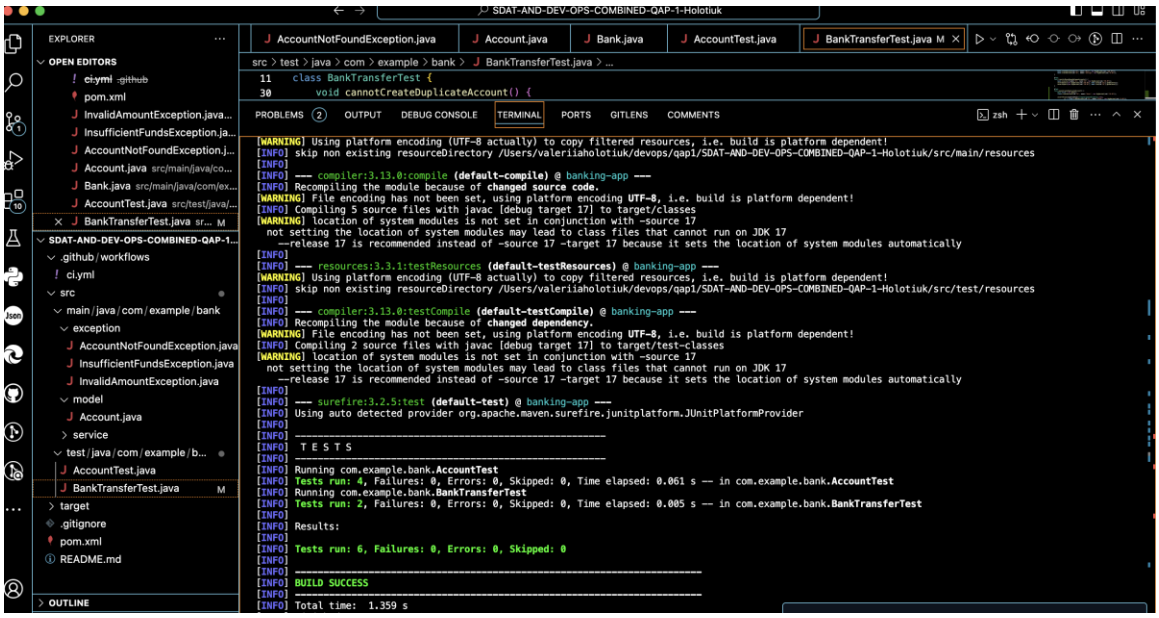
- Account.java: Represents a bank account, enforces rules for deposits/withdrawals.
- Bank.java: Manages accounts, prevents duplicates, handles transfers.
- Exceptions:
  - InvalidAmountException (deposit/withdraw invalid amounts)
  - InsufficientFundsException (not enough balance)
  - AccountNotFoundException (invalid account ID)

3. Unit Testing

Tests are written using JUnit 5 to cover both positive and negative scenarios.

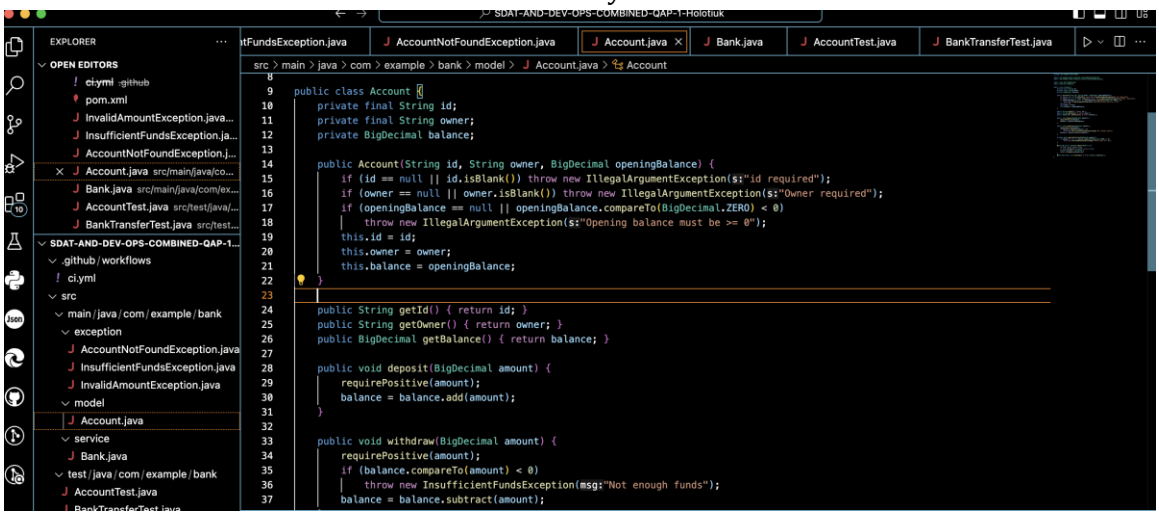
- AccountTest
  - Deposit increases balance (positive)
  - Withdraw decreases balance (positive)
  - Withdraw more than balance → throws InsufficientFundsException (negative)
  - Deposit zero or negative → throws InvalidAmountException (negative)

- BankTransferTest
  - Transfer moves money between accounts (positive)
  - Duplicate account creation → throws IllegalArgumentException (negative)



4. Clean Code Practices

1. Fail-fast validation
   - Constructor checks invalid values immediately.



2. Separation of Concerns
   - Account handles account logic.
   - Bank handles acco unt management.

```java
public class Bank {
    private final Map<String, Account> accounts = new HashMap<>();

    public Account createAccount(String id, String owner, BigDecimal openingBalance) {
        if (accounts.containsKey(id)) throw new IllegalArgumentException("Account already exists: " + id);
        Account acc = new Account(id, owner, openingBalance);
        accounts.put(id, acc);
        return acc;
```

3. Explicit exceptions
  - Custom exceptions make errors descriptive and easy to debug.

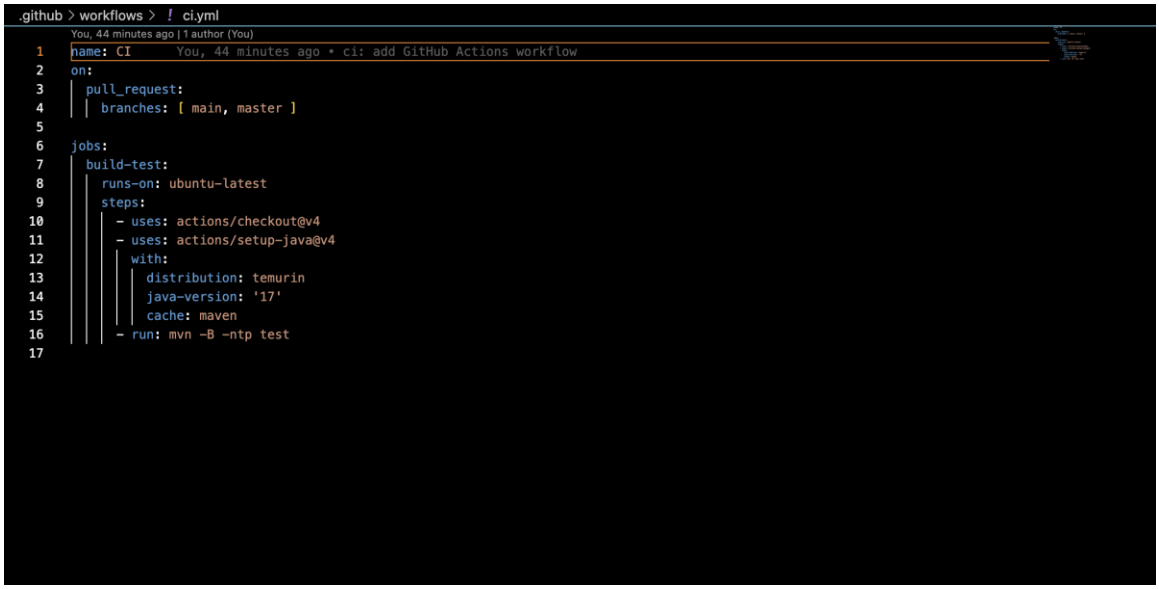
5. Dependencies

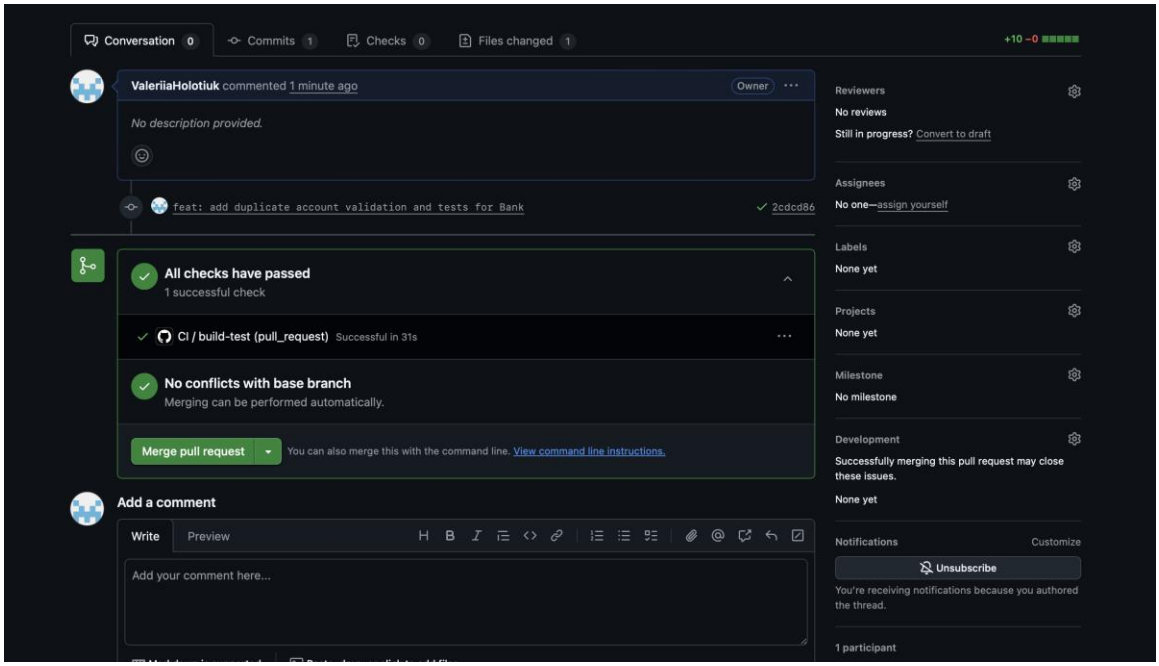
Dependencies managed in pom.xml:
- JUnit 5 (org.junit.jupiter:junit-jupiter:5.10.2)
- Maven Surefire Plugin (3.2.5)

All dependencies were retrieved from Maven Central Repository.

6. Continuous Integration (CI)

GitHub Actions workflow (.github/workflows/ci.yml) automatically runs mvn test on every Pull Request.



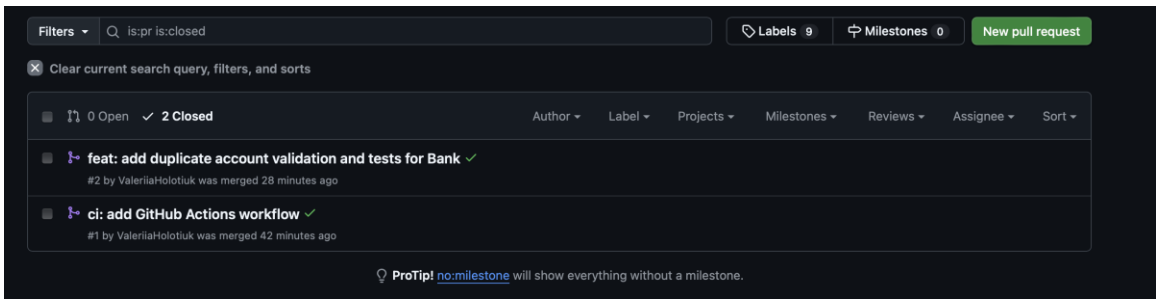


7. Problems & Fixes

- Wrong Java syntax (val:"40.00") → fixed to new BigDecimal("40.00").
- GitHub Actions file in wrong folder → moved to .github/workflows/ci.yml.
- Added duplicate account validation + test.

8. Trunk-Based Workflow Evidence

- Branch chore/ci-setup → added CI workflow → PR → merged.
- Branch test/duplicate-account → added negative test → PR → merged.



9. Conclusion

This project demonstrates:
- Object-oriented design in Java
- Comprehensive unit testing with JUnit 5
- Build management using Maven
- Continuous Integration using GitHub Actions
- Clean code practices
- Trunk-based development workflow