Data Flow Testing for UserManager.signup Method

We'll start by creating a Control Flow Graph (CFG) for the UserManager.signup method, identifying the nodes and edges, and then proceed with defining the necessary criteria for data flow testing.

Explanation of CFG Nodes and Edges

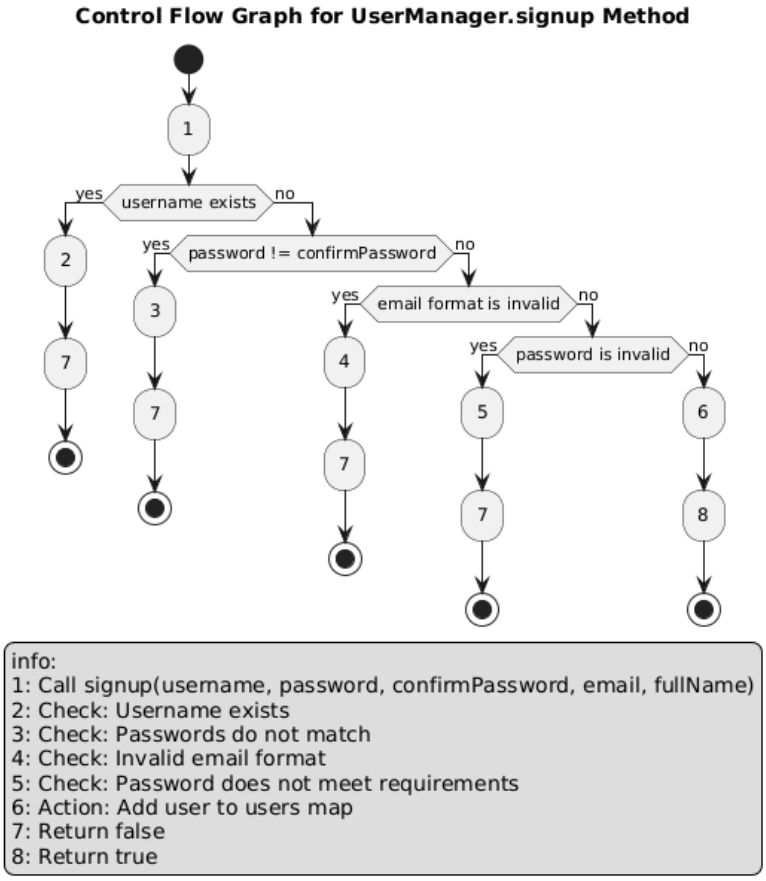
Nodes:

1. **Start: Call signup(username, password, confirmPassword, email, fullName)**
2. **Check: Username exists**
3. **Check: Passwords do not match**
4. **Check: Invalid email format**
5. **Check: Password does not meet requirements**
6. **Action: Add user to users map**
7. **Return false**
8. **Return true**

Edges:

- Edge 1: From node 1 to 2
- Edge 2: From node 2 to 7
- Edge 3: From node 2 to 3
- Edge 4: From node 3 to 7
- Edge 5: From node 3 to 4
- Edge 6: From node 4 to 7
- Edge 7: From node 4 to 5
- Edge 8: From node 5 to 7
- Edge 9: From node 5 to 6
- Edge 10: From node 6 to 8

CFG



Identifying Defs and Uses

- **Defs**: username, password, confirmPassword, email, fullName
- **Uses**: username (2), password (3, 5), confirmPassword (3), email (4), fullName (6)

DU Pairs and DU Paths

- DU Pair for username: (1, 2)
- DU Pair for password: (1, 3), (1, 5)
- DU Pair for confirmPassword: (1, 3)
- DU Pair for email: (1, 4)
- DU Pair for fullName: (1, 6)

Defining Criteria

- **All-defs Coverage (ADC)**: Ensure that each definition of a variable is covered by at least one use.
- **All-uses Coverage (AUC)**: Ensure that all uses of variables are covered by paths from their definitions.
- **All-du-paths Coverage (ADUPC)**: Ensure that all definition-use paths are covered.

DU Paths and Paths Coverage

DU Paths for username:

1. Path 1: 1 -> 2
2. Path 2: 1 -> 3 -> 4 -> 5 -> 6 -> 8

DU Paths for password:

1. Path 1: 1 -> 3
2. Path 2: 1 -> 5

DU Paths for confirmPassword:

1. Path 1: 1 -> 3

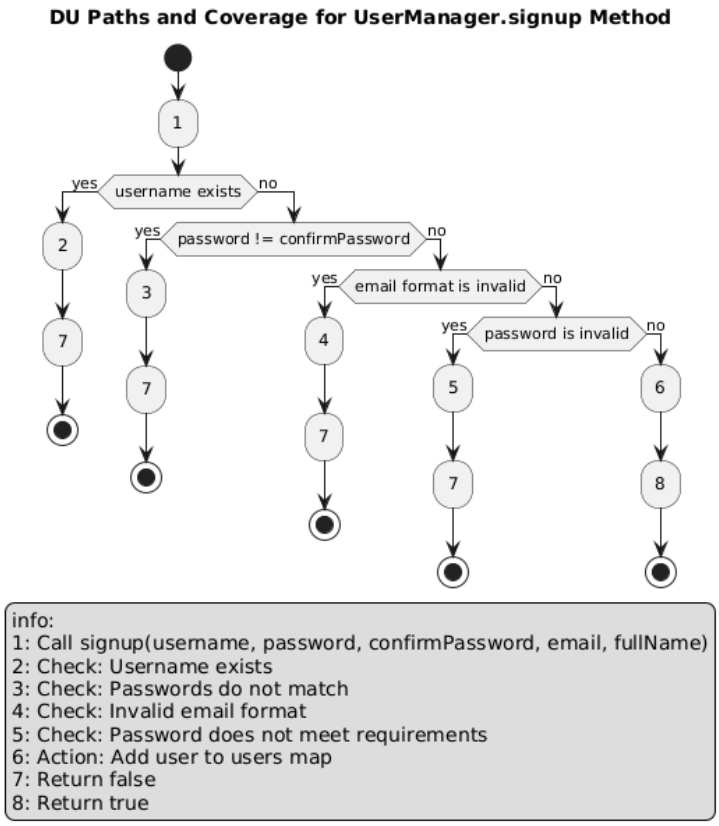
DU Paths for email:

1. Path 1: 1 -> 4

DU Paths for fullName:

1. Path 1: 1 -> 6

DU Paths and Coverage Graph



Testing Paths Coverage

1. **ADC Paths**:
    - username: 1 -> 2
    - password: 1 -> 3, 1 -> 5
    - confirmPassword: 1 -> 3
    - email: 1 -> 4
    - fullName: 1 -> 6
2. **AUC Paths**:
    - username: 1 -> 2
    - password: 1 -> 3, 1 -> 5
    - confirmPassword: 1 -> 3
    - email: 1 -> 4
    - fullName: 1 -> 6
3. **ADUPC Paths**:
    - username: 1 -> 2, 1 -> 3 -> 4 -> 5 -> 6 -> 8
    - password: 1 -> 3, 1 -> 5
    - confirmPassword: 1 -> 3
    - email: 1 -> 4
    - fullName: 1 -> 6

These paths ensure all uses of the variables from their definitions are covered.

Now, to implement data flow testing in Java, we can write JUnit tests that specifically target the definition-use (DU) pairs in the signup method. The goal is to cover all definitions and uses of variables in the method. We'll focus on testing the paths that include these DU pairs.

JUnit Test for Data Flow Testing
Here is the detailed implementation of JUnit tests to cover all DU pairs for the signup method in the UserManager class.

UserManager.java
Let's start by confirming the UserManager class for reference:

```
import java.util.HashMap;
import java.util.Map;

public class UserManager {
  private Map<String, User> users = new HashMap<>();

  public boolean signup(String username, String password, String confirmPassword, String email, String fullName) {
    if (users.containsKey(username)) {
      System.out.println("Username already exists.");
      return false;
    }
    if (!password.equals(confirmPassword)) {
      System.out.println("Passwords do not match.");
      return false;
    }
    if (!isValidEmail(email)) {
      System.out.println("Invalid email format.");
      return false;
    }
    if (!isValidPassword(password)) {
      System.out.println("Password must be at least 8 characters long, and include uppercase, lowercase, numbers, and symbols.");
      return false;
    }
    users.put(username, new User(username, password, email, fullName));
    return true;
  }

  public User signin(String username, String password) {
    User user = users.get(username);
```

```
        if (user != null && user.getPassword().equals(password)) {
            return user;
        }
        return null;
    }

    public Map<String, User> getAllUsers() {
        return new HashMap<>(users);
    }

    private boolean isValidEmail(String email) {
        return email.matches("^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$");
    }

    private boolean isValidPassword(String password) {
        return password.matches("^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[@$!%*?&])[A-Za-
z\\d@$!%*?&]{8,}$");
    }
}
```

JUnit Tests

Now, we will write the JUnit tests to cover all DU pairs.

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class UserManagerTest {

    private UserManager userManager;

    @BeforeEach
    void setUp() {
        userManager = new UserManager();
    }

    @Test
    void testSignup_UsernameExists() {
        userManager.signup("existingUser", "Password123!", "Password123!",
"existing@user.com", "Existing User");
        boolean result = userManager.signup("existingUser", "Password123!", "Password123!",
"new@user.com", "New User");
        assertFalse(result);
    }
```

```
  @Test
  void testSignup_PasswordsDoNotMatch() {
    boolean result = userManager.signup("newUser", "Password123!", "Password1234!",
"new@user.com", "New User");
    assertFalse(result);
  }

  @Test
  void testSignup_InvalidEmail() {
    boolean result = userManager.signup("newUser", "Password123!", "Password123!",
"invalid-email", "New User");
    assertFalse(result);
  }

  @Test
  void testSignup_InvalidPassword() {
    boolean result = userManager.signup("newUser", "password", "password",
"new@user.com", "New User");
    assertFalse(result);
  }

  @Test
  void testSignup_Success() {
    boolean result = userManager.signup("newUser", "Password123!", "Password123!",
"new@user.com", "New User");
    assertTrue(result);
  }

  @Test
  void testSignup_AllDUPaths() {
    // Test 1: Definition and use of username (Path: 1 -> 2 -> 7)
    userManager.signup("existingUser", "Password123!", "Password123!",
"existing@user.com", "Existing User");
    boolean result1 = userManager.signup("existingUser", "Password123!", "Password123!",
"new@user.com", "New User");
    assertFalse(result1);

    // Test 2: Definition and use of password and confirmPassword (Path: 1 -> 3 -> 7)
    boolean result2 = userManager.signup("newUser", "Password123!", "Password1234!",
"new@user.com", "New User");
    assertFalse(result2);

    // Test 3: Definition and use of email (Path: 1 -> 4 -> 7)
```

```
        boolean result3 = userManager.signup("newUser", "Password123!", "Password123!",
"invalid-email", "New User");
        assertFalse(result3);

        // Test 4: Definition and use of password (Path: 1 -> 5 -> 7)
        boolean result4 = userManager.signup("newUser", "password", "password",
"new@user.com", "New User");
        assertFalse(result4);

        // Test 5: Definition and use of all parameters for success (Path: 1 -> 6 -> 8)
        boolean result5 = userManager.signup("newUser", "Password123!", "Password123!",
"new@user.com", "New User");
        assertTrue(result5);
    }
}
```

These tests should provide comprehensive coverage of the signup method, ensuring all DU pairs are tested.