To create test cases for Boundary Value Analysis (BVA) based on the requirements in your document for an automated banking application, I'll provide a Java code template using JUnit in Eclipse Oxygen IDE. The test cases are designed around the boundary conditions for:

1. **Area Code -** blank or a three-digit number
2. **Prefix -** a three-digit number, not beginning with 0 or 1
3. **Suffix -** a four-digit number
4. **Password -** six-character alphanumeric
5. **Commands -** "Check status", "Deposit", or "Withdrawal"

Here are the test cases:

Positive and Negative Boundary Value Analysis Test Cases

Positive Test Cases

1. Area Code: Minimum (000), Prefix: 200, Suffix: 1000, Password: 6 alphanumeric characters, Command: "Check status"
2. Area Code: Maximum (999), Prefix: 999, Suffix: 9999, Password: 6 alphanumeric characters, Command: "Deposit"
3. Prefix: Minimum (200), Suffix: Minimum (1000)
4. Prefix: Maximum (999), Suffix: Maximum (9999)
5. Password: Minimum valid length (6 characters)
6. Command: "Check status" (valid command)
7. Command: "Deposit" (valid command)
8. Command: "Withdrawal" (valid command)
9. Area Code: Valid three digits (123)
10. Suffix: Mid-value (5000)

Negative Test Cases

1. Area Code: Below minimum (-1)
2. Area Code: Above maximum (1000)
3. Prefix: Below minimum (199)
4. Prefix: Above maximum (1000)
5. Suffix: Below minimum (999)
6. Suffix: Above maximum (10000)
7. Password: Below minimum (5 characters)
8. Password: Above maximum (7 characters)
9. Command: Invalid command ("Transfer")
10. Area Code: Non-numeric (ABC)

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

public class BankingApplicationTest {

   // Helper method to simulate validation (for demonstration)
   public boolean validateInput(String areaCode, String prefix, String suffix, String password, String
command) {
       return areaCode.matches("\\d{3}") && Integer.parseInt(prefix) >= 200 &&
Integer.parseInt(prefix) <= 999 &&
              suffix.matches("\\d{4}") && password.matches("\\w{6}") &&
              (command.equals("Check status") || command.equals("Deposit") ||
command.equals("Withdrawal"));
   }

   // Positive Test Cases
   @Test
   public void testValidMinAreaCode() {
       assertTrue(validateInput("000", "200", "1000", "abc123", "Check status"));
   }

   @Test
   public void testValidMaxAreaCode() {
       assertTrue(validateInput("999", "999", "9999", "xyz456", "Deposit"));
   }

   @Test
   public void testValidMinPrefix() {
       assertTrue(validateInput("123", "200", "1000", "pass12", "Withdrawal"));
   }

   @Test
   public void testValidMaxPrefix() {
       assertTrue(validateInput("456", "999", "9999", "code34", "Check status"));
   }

   @Test
   public void testValidPasswordLength() {
       assertTrue(validateInput("789", "300", "2000", "abcdef", "Deposit"));
   }

   @Test
   public void testValidCommandCheckStatus() {
       assertTrue(validateInput("222", "500", "3500", "123abc", "Check status"));
   }

   @Test
   public void testValidCommandDeposit() {
       assertTrue(validateInput("333", "600", "4500", "xyz123", "Deposit"));
   }

   @Test
   public void testValidCommandWithdrawal() {
       assertTrue(validateInput("444", "700", "5500", "pass45", "Withdrawal"));
   }

   @Test
   public void testValidAreaCode() {
       assertTrue(validateInput("123", "250", "2500", "xyz789", "Deposit"));
   }

   @Test
   public void testValidSuffixMidValue() {
       assertTrue(validateInput("555", "350", "5000", "abc456", "Withdrawal"));
   }
```

```
// Negative Test Cases
@Test
public void testInvalidAreaCodeBelowMin() {
    assertFalse(validateInput("-1", "200", "1000", "abc123", "Check status"));
}

@Test
public void testInvalidAreaCodeAboveMax() {
    assertFalse(validateInput("1000", "300", "2000", "xyz456", "Deposit"));
}

@Test
public void testInvalidPrefixBelowMin() {
    assertFalse(validateInput("111", "199", "3000", "abc789", "Withdrawal"));
}

@Test
public void testInvalidPrefixAboveMax() {
    assertFalse(validateInput("222", "1000", "4000", "pass12", "Check status"));
}

@Test
public void testInvalidSuffixBelowMin() {
    assertFalse(validateInput("333", "400", "0999", "code45", "Deposit"));
}

@Test
public void testInvalidSuffixAboveMax() {
    assertFalse(validateInput("444", "500", "10000", "xyz123", "Withdrawal"));
}
```

```
    @Test
    public void testInvalidPasswordLengthBelowMin() {
        assertFalse(validateInput("555", "600", "1100", "abcd1", "Check status"));
    }

    @Test
    public void testInvalidPasswordLengthAboveMax() {
        assertFalse(validateInput("666", "700", "2200", "abcdef1", "Deposit"));
    }

    @Test
    public void testInvalidCommand() {
        assertFalse(validateInput("777", "800", "3300", "xyz789", "Transfer"));
    }

    @Test
    public void testInvalidAreaCodeNonNumeric() {
        assertFalse(validateInput("ABC", "900", "4400", "abc123", "Withdrawal"));
    }
}
```