# 13.JUnitTestforVotingSystem(White-BoxTesting)

Aim:

To create a JUnit test for a voting system, using white-box testing techniques. This will involveverifyingthesystem'sfunctionalitybytestingindividualmethodsandlogicpathsin the system.

Algorithm:

1. **Step1:**Understandthecorefunctionalityofthevotingsystem(e.g.,checking eligibility, casting a vote, and counting votes).
2. **Step2:**Designamethod forcheckingvotereligibility(e.g.,agegreaterthanorequal to 18).
3. **Step3:**Createamethod tocastavote,whereonlyeligiblevoterscanvote.
4. **Step4:**WriteaJUnittestclasstoverifythesystem'sfunctionalitybytestingdifferent scenarios:
   * Votereligibility
   * Votecasting
   * Votecounting
5. **Step5:**Usewhite-boxtestingtoensurethateachindividuallogicpath(suchasvalid or invalid age checks) is thoroughly tested.

Code:

**package** ex5.sub;

**import** java.util.HashMap;

**import** java.util.Map;

**publicclass** Voting {

**privatefinal** Map<String, Integer>voteCount = **new** HashMap<>();

**privatefinal** String[] validCandidates = {"A", "B"};

**publicboolean**checkEligibility(**int**age) {

**return**age>= 18;

}

**publicboolean** castVote(**int**age, String candidate) {

**if** (!checkEligibility(age)) {

**returnfalse**;

}

**for** (String validCandidate : validCandidates) {

**if** (validCandidate.equals(candidate)) {

voteCount.put(candidate, voteCount.getOrDefault(candidate, 0) + 1);

**returntrue**;

}

}

**returnfalse**;

}

**publicint** countVotes(String candidate) {

**return**voteCount.getOrDefault(candidate, 0);

}

}

*JUnitTestClass*

**package** ex5.sub;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.Test;

**importstatic** org.junit.jupiter.api.Assertions.\*;

**class** VotingTest {

**private** Voting votingSystem;

@BeforeEach

**publicvoid** setUp() {

votingSystem = **new** Voting();

}

@Test

**publicvoid** testCheckEligibility() {

*assertTrue*(votingSystem.checkEligibility(18)); // Test Case 1

}

@Test

**publicvoid** testCastVoteEligibleValidCandidate() {

*assertTrue*(votingSystem.castVote(20, "A")); // Test Case 2

}

@Test

**publicvoid** testCastVoteUnderage() {

*assertFalse*(votingSystem.castVote(16, "A")); // Test Case 3

}

@Test

**publicvoid** testCastVoteInvalidCandidate() {

*assertFalse*(votingSystem.castVote(20, "C")); // Test Case 4

}

@Test

**publicvoid**testCountVotesForCandidateA() {

votingSystem.castVote(20, "A");

votingSystem.castVote(25, "A");

*assertEquals*(2, votingSystem.countVotes("A")); // Test Case 5

}

}

Sample Input:

# TestCase1:

* + Input:checkEligibility(18)
  + ExpectedOutput: true(Eligibleto vote)

# TestCase2:

* + Input:castVote(20, "A")
  + ExpectedOutput:true(VotecastsuccessfullyforcandidateA)

# TestCase3:

* + Input:castVote(16, "A")
  + ExpectedOutput:false(Underagevoter, cannotvote)

# TestCase4:

* + Input:castVote(20, "C")
  + ExpectedOutput:false(Invalid candidate)

# TestCase5:

* + Input:countVotes("A")
  + ExpectedOutput:2(TwovotesforcandidateA) Sample Output:

# TestCase1:

* + Input:checkEligibility(18)
  + Output:true

# TestCase2:

* + Input:castVote(20, "A")
  + Output:true

# TestCase3:

* + Input:castVote(16, "A")
  + Output:false

# TestCase4:

* + Input:castVote(20, "C")
  + Output:false

# TestCase5:

* + Input:countVotes("A")
  + Output:2

Results:

* **TestCase1:**ThecheckEligibilitymethodsuccessfullyvalidatedtheeligibilityof voters based on age.
* **TestCase2:**ThecastVotemethodsuccessfullycastvotesforeligiblevotersand rejected votes for underage voters.
* **TestCase3:**Themethodcorrectlyrejectedvotesfromunderageusers.
* **TestCase4:**ThecastVotemethodcorrectlyrejectedinvalidcandidates.
* **TestCase5:**ThecountVotesmethodcorrectlycountedvotesforvalidcandidatesand
* returned 0 for invalid candidates.



