To combat election fraud, your city is instituting a new voting procedure. The ballot has a letter associated with every selection a voter may make. A sample ballot is shown.

1. VOTE FOTR MAYOR

A. Pincher, Penny 🞎

B. Dover, Skip 🞎

C. Perman, Sue 🞎

2. PROPOSITION 17

D. YES 🞎

E. NO 🞎

3. MEASURE 1

F. YES 🞎

G. NO 🞎

4. MEASURE 2

H. YES 🞎

I. NO 🞎

After submitting their ballot, every voter receives a receipt that has a unique ID number and a record of their voting selections. For example, a voter that submits a ballot for Sue Perman, Yes on Proposition 17, No on Measure 1, and Yes on Measure 2 might receive a receipt with

ID 4925 : CDGH

The next day, the city posts all votes on their web page sorted by ID number. This allows a voter to confirm their submission and allows anyone to count the vote totals for themselves. A sample list for the sample ballot is shown.

ID VOTES

4925 CDGH

4926 AEGH

4927 CDGI

4928 BEGI

4929 ADFH

Write a program that reads the posted voting list from a file and outputs the percent of votes cast for each ballot item. You may assume that the file does not have any header lines. Each line will contain a voter ID and a string representing votes. Define a class named Voter that stores an individual’s voting record. The class should have a constructor that takes as input a string of votes (e.g. “CDGH”) and a voter ID. It should have accessors for the voter ID and the vote for a specific question.

# Application Development – TDD

* Create the solution
* Create the VoterFraudModel class library project
* Create the VoterFraudModel Developer Testing class library project
* Install NUnit
* Set up project references
* Write the first test

|  |
| --- |
| [TestFixture]  public class AVoter  {  [Test]  public void ShouldStoreAndReportTheVotersID()  {  var sut = new Voter("4925");  Assert.That(sut.ID, Is.EqualTo("4925"));  }  } |

* Write the code to fix the compiler errors:

|  |
| --- |
| public class Voter  {  public string ID { get; set; }  public Voter(string id)  {    }  } |

* The test should fail
* Add the code to pass the test

|  |
| --- |
| public class Voter  {  public string ID { get; set; }  public Voter(string id)  {  ID = id;  }  } |

* Write the second test:

|  |
| --- |
| [Test]  public void ShouldStoreAndReportTheVoteForQuestion1()  {  var sut = new Voter("4925", "CDGH");  Assert.That(sut.Vote(1), Is.EqualTo('C'));  } |

* Fix the compiler errors:

|  |
| --- |
| public class Voter  {  public string ID { get; set; }  public Voter(string id, string vote)  {  ID = id;  }  public char Vote(int questionNumber)  {  return ' ';  }  } |
| [Test]  public void ShouldStoreAndReportTheVotersID()  {  var sut = new Voter("4925", "Don't care");  Assert.That(sut.ID, Is.EqualTo("4925"));  } |

* The test should fail
* Write code to pass the test

|  |
| --- |
| public class Voter  {  public string ID { get; set; }  private string \_vote;  public Voter(string id, string vote)  {  ID = id;  \_vote = vote;  }  public char Vote(int questionNumber)  {  return \_vote[questionNumber-1];  }  } |

* Create tests for the other cases and refactor the test case:

|  |
| --- |
| [TestCase(1, 'C')]  [TestCase(2, 'D')]  [TestCase(3, 'G')]  [TestCase(4, 'H')]  public void ShouldStoreAndReportTheVoteForQuestion1To4(int questionNumber, char expected)  {  var sut = new Voter("4925", "CDGH");  Assert.That(sut.Vote(questionNumber), Is.EqualTo(expected));  } |

* Create first voting list integration test

|  |
| --- |
| public void ShouldReportCorrectPercentageVotesForOneVoter()  {  var sut = new VoterList();  sut.AddVoter(new Voter("4925", "CDGH"));  Assert.That(sut.PercentVote('A'), Is.EqualTo(0));  Assert.That(sut.PercentVote('B'), Is.EqualTo(0));  Assert.That(sut.PercentVote('C'), Is.EqualTo(100));  Assert.That(sut.PercentVote('D'), Is.EqualTo(100));  Assert.That(sut.PercentVote('E'), Is.EqualTo(0));  Assert.That(sut.PercentVote('F'), Is.EqualTo(0));  Assert.That(sut.PercentVote('G'), Is.EqualTo(100));  Assert.That(sut.PercentVote('H'), Is.EqualTo(100));  Assert.That(sut.PercentVote('I'), Is.EqualTo(0));  } |

* Fix compiler errors

|  |
| --- |
| public class VoterList  {  public void AddVoter(Voter voter)  {  }  public decimal PercentVote(char vote)  {  return 0.0m;  }  } |

* Install Moq
* Refactor to create an interface for Voter

|  |
| --- |
| public interface IVoter  {  string ID { get; set; }  char Vote(int questionNumber);  }  public class Voter : IVoter  … |

* Modify VoterList to use the interface:

|  |
| --- |
| public void AddVoter(IVoter voter)  {  } |

* The new test should fail
* Add the necessary code to make the test pass

|  |
| --- |
| public class VoterList  {  private ICollection<IVoter> \_voters;  public VoterList(ICollection<IVoter> voters = null)  {  if(voters == null)  {  voters = new List<IVoter>();  }  \_voters = voters;  }  public void AddVoter(IVoter voter)  {  \_voters.Add(voter);  }  public decimal PercentVote(char vote)  {  var voterCount = 0;  foreach(var voter in \_voters)  {  for(var question = 1; question <= 4; question++)  {  if (voter.Vote(question) == vote)  {  voterCount++;  }  }  }  return (voterCount / (decimal)\_voters.Count)\*100.0m;  }  } |

* The tests should all pass
* Refactor:

|  |
| --- |
| public class VoterList  {  private ICollection<IVoter> \_voters;  private readonly int \_numberOfQuestions;  public VoterList(int numberOfQuestions = 4, ICollection<IVoter> voters = null)  {  if(voters == null)  {  voters = new List<IVoter>();  }  \_voters = voters;  \_numberOfQuestions = numberOfQuestions;  }  public void AddVoter(IVoter voter)  {  \_voters.Add(voter);  }  public decimal PercentVote(char vote)  {  var voterCount = 0;  foreach(var voter in \_voters)  {  for(var question = 1; question <= \_numberOfQuestions; question++)  {  if (voter.Vote(question) == vote)  {  voterCount++;  }  }  }  return (voterCount / (decimal)\_voters.Count)\*100.0m;  }  } |

* Add second test:

|  |
| --- |
| [Test]  public void ShouldReportCorrectPercentageVotesForFiveVoters()  {  var sut = new VoterList();  sut.AddVoter(new Voter("4925", "CDGH"));  sut.AddVoter(new Voter("4926", "AEGH"));  sut.AddVoter(new Voter("4927", "CDGI"));  sut.AddVoter(new Voter("4928", "BEGI"));  sut.AddVoter(new Voter("4929", "ADFH"));  Assert.That(sut.PercentVote('A'), Is.EqualTo(40));  Assert.That(sut.PercentVote('B'), Is.EqualTo(20));  Assert.That(sut.PercentVote('C'), Is.EqualTo(40));  Assert.That(sut.PercentVote('D'), Is.EqualTo(60));  Assert.That(sut.PercentVote('E'), Is.EqualTo(40));  Assert.That(sut.PercentVote('F'), Is.EqualTo(20));  Assert.That(sut.PercentVote('G'), Is.EqualTo(80));  Assert.That(sut.PercentVote('H'), Is.EqualTo(60));  Assert.That(sut.PercentVote('I'), Is.EqualTo(40));  } |

* Add the next test:

|  |
| --- |
| [Test]  public void ShouldBeAbleToReadVotersFromTextFile()  {  var sut = new VoterList();  sut.ReadVotersFromFile(new StreamReader(@"**PATH\**voterlist.txt"));  Assert.That(sut.PercentVote('A'), Is.EqualTo(40));  Assert.That(sut.PercentVote('B'), Is.EqualTo(20));  Assert.That(sut.PercentVote('C'), Is.EqualTo(40));  Assert.That(sut.PercentVote('D'), Is.EqualTo(60));  Assert.That(sut.PercentVote('E'), Is.EqualTo(40));  Assert.That(sut.PercentVote('F'), Is.EqualTo(20));  Assert.That(sut.PercentVote('G'), Is.EqualTo(80));  Assert.That(sut.PercentVote('H'), Is.EqualTo(60));  Assert.That(sut.PercentVote('I'), Is.EqualTo(40));  } |

* Fix the compile errors:

|  |
| --- |
| public void ReadVotersFromFile(StreamReader streamReader)  {    } |

* The test should fail
* Make the test pass:

|  |
| --- |
| public void ReadVotersFromFile(StreamReader streamReader)  {  string line;  while (streamReader.Peek() >= 0)  {  line = streamReader.ReadLine();  var record = line.Split(' ');  \_voters.Add(new Voter(record[0], record[1]));  }  } |