Suppose that you are creating a fantasy role-playing game. In this game there are four types of creatures: humans, cyberdemons, balrogs, and elves. To represent these creatures, we might define a Creature class as follows:

|  |
| --- |
| public class Creature  {  /// <summary>  /// 0 = Human, 1 = Cyberdemon, 2 = Balrog, 3 = Elf  /// </summary>  public int Type { get; set; }  public int Strength { get; set; }  public int HitPoints { get; set; }  public string Race  {  get  {  switch (Type)  {  case 0: return "Human";  case 1: return "Cyberdemon";  case 2: return "Balrog";  case 3: return "Elf";  }  return "Unknown";  }  }  private Random \_random;  public Creature()  {  \_random = new Random();  Type = 0; // Human  Strength = 50;  HitPoints = 100;  }  public int TakeDamage(int damage)  {  // All creatures have a 1% chance of dodging the damage  if (\_random.Next(100) < 1)  {  damage = 0;  }  HitPoints -= damage;  return damage;  }  public int CalculateDamage()  {  int damage = 0;  // All creatures inflict damage that is a random number up to their strength  // with a minimum damage of 1  damage = \_random.Next(Strength) + 1;  // Humans are observant, they have a 10% chance of spotting a  // weakness and can inflict double damage  if (Type == 0)  {  if (\_random.Next(100) < 10)  {  damage \*= 2;  }  }  // Demons have a 25% chance of inflicting extra 10 damage  else if ((Type == 2) || (Type == 1))  {  if (\_random.Next(100) < 25)  {  damage += 10;  }  }  // Elves have a 10% chance of inflicting additional magical damage  // which is at half strength  else if (Type == 3)  {  if (\_random.Next(100) < 10)  {  damage += \_random.Next(Strength/2) + 1;  }  }  // Balrogs are deadly and can inflict base damage twice  if (Type == 2)  {  damage += \_random.Next(Strength) + 1;  }  return damage;  }  public int Attack(Creature creature)  {  return creature.TakeDamage(CalculateDamage());  }  } |

One problem with this implementation is that it is unwieldy to add new creatures.

# 1: Create the Projects

1. Create solution ***FantasyRPGGame***
   1. Class Library Project: ***FantasyRPGGame.Model***
   2. NUnit Project: ***FantasyRPGGame.UnitTesting***
   3. NUnit Project: ***FantasyRPGGame.IntegrationTesting***
   4. Install testing packages:  
      Install-Package NUnit  
      Install-Package NUnit3TestAdapter  
      Install-Package Microsoft.NET.Test.Sdk  
      Install-Package Moq
2. **Redesign** the Creature class to eliminate the need for the Type property
   1. The Creature class should be abstract
   2. Create an abstraction for the Random dependency
      1. Inject it into the Creature class and make it protected
   3. Race is an abstract get property
   4. All methods and properties are virtual
   5. CalculateDamage
      1. All creatures inflict damage that is a random number up to their strength with a minimum damage of 1
3. Unit test the Creature class
   1. Focus on the TakeDamage method
   2. Since Creature is abstract, create a DummyCreature class derived from Creature in the testing project to substitute
   3. You will have to fake the Creature class and mock the random abstraction
4. **Milestone 1**
   1. **Categorize each test as Milestone 1**
   2. **Show the class and unit tests to the instructor**

# 2: Humans

1. Implement the Human class, which is derived from Creature
   1. Race is “Human”
   2. 10% chance of inflicting double damage
2. Unit test the Human class
3. **Milestone 2**
   1. **Categorize each test as Milestone 2**
   2. **Show the class and unit tests to the instructor**

# 3: Integration Test - Creature

1. Create integration tests for the Creature class and your *Random* concretion. Here are the tests to write. Think outside the box for how to test them.
   1. A creature has 99% chance of taking damage
   2. A create has 1% change of not taking damage
2. **Milestone 3**
   1. **Categorize each test as Milestone 3**
   2. **Show the integration tests to the instructor**

# 4: Demons

1. Implement an abstract Demon class derived from Creature
   1. Override and implement the CalculateDamage method
      1. Demons have a 25% chance of inflicting extra 10 damage
2. Unit test the Demon class
   1. A demon has 25% chance of inflicting additional 10 damage
   2. A demon has 75% chance of inflicting base damage
3. **Milestone 4**
   1. **Categorize each test as Milestone 4**
   2. **Show the integration tests to the instructor**

# 5: Balrogs

1. Implement the Balrog class derived from Demon
   1. Override and implement the Race property
   2. Override and implement the CalculateDamage method
      1. Balrogs are deadly and can inflict base damage twice
2. Unit test the Balrog class
   1. A Balrog should report its race as “Balrog”
   2. A Balrog inflicts base damage twice
3. **Milestone 5**
   1. **Categorize each test as Milestone 5**
   2. **Show the unit tests to the instructor**

# 6: Battle

1. Create a class called Battle
   1. Private collection of Creatures as a List
   2. Public collection of strings called Messages
2. Add a method to add a Creature to the collection
3. Add a method to return a Creature at a given index
4. Add a method to remove a Creature at a given index
5. Add a method called Duel
   1. Two parameters: index1 and index2, which represents the index of the two creatures that will duel
   2. Returns the index of the creature that won the duel or -1 if the duel is a tie
   3. Each creature will repeatedly attack each until the hit points of one fall below 1
      1. A creature loses if its hit points fall below 1 first
      2. The duel is a tie if both creatures fall below 1 hit points
      3. The hit points are resolved after each creature attacks the other
   4. During the duel, the result of each attack is stored into Messages
      1. e.g. “*The Balrog deals 10 damage to the Human.*”
   5. After the duel, a result message is posted into Messages
6. Unit test the Battle class
   1. Focus on the Duel method
   2. Use the following class perform the unit tests

|  |
| --- |
| public class TestCreature : Creature  {  public int Damage { get; set; }  public TestCreature(IRandom random = null) : base(random)  {  }  public override string Race => "TestCreature";  public override int Attack(Creature creature)  {  return creature.TakeDamage(Damage);  }  } |

* 1. Add the following tests
     1. ShouldReportThatCreature1WonTheDuel
     2. ShouldReportThatCreature2WonTheDuel
     3. ShouldReportATie

1. **Milestone 6**
   1. **Categorize each test as Milestone 6**
   2. **Show the unit tests to the instructor**

# 7: Console App

1. Write a console application to report the results of a Human dueling a Balrog. Repeat the duel 100 times and then report the number of times the human wins, the Balrog wins, and the number of ties. Also, report the message of each attack.
2. **Milestone 7**
   1. **Show the console application to the instructor**