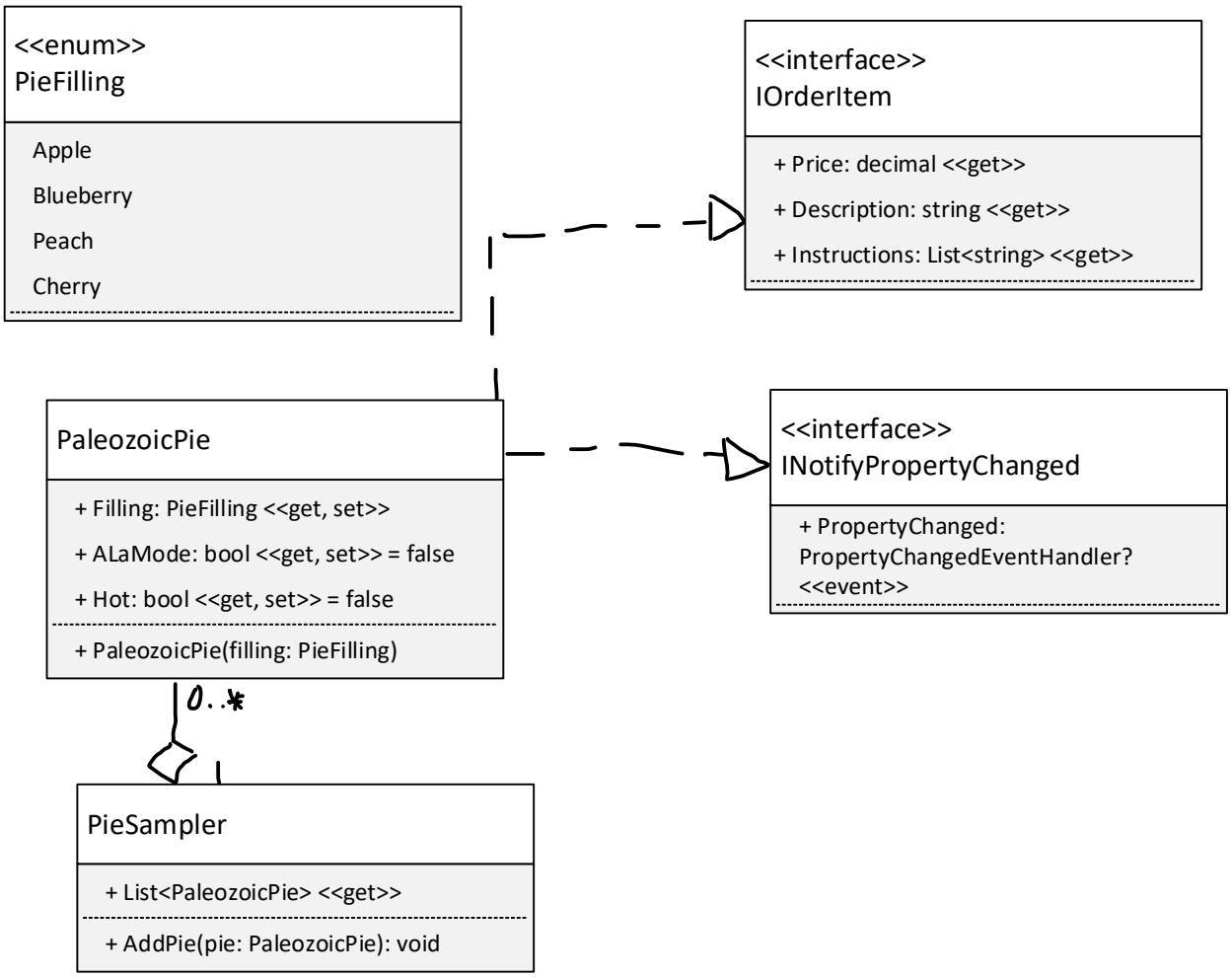


1. Add the appropriate UML associations to the diagram above. We want to indicate that *PaleozoicPie* implements *INotifyPropertyChanged* and *IOrderItem*.

2. Write the *PaleozoicPie* class definition as modeled by the UML diagram above (assuming the correct associations were added in #1). The *Description* should be the filling followed by "Paleozoic Pie." A Paleozoic Pie is $3.00 plus an additional $1 when served with ice cream (a la mode). The *Instructions* property should include "Hot" if the pie is hot and "A La Mode" if the pie is served a la mode.

```
public class PaleozoicPie: INotifyPropertyChanged, IOrderItem {
        public event PropertyChangedEventHandler? PropertyChanged;

        private PieFilling _filling;
        public PieFilling Filling {
                get => _filling;
                set {
                        _filling = value;
                        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Filling"));
                        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Description"));

                }
        }

        private bool _alaMode = false;
        public bool ALaMode {
                get => _alaMode;
                set {
                        _alaMode = value;
                        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("ALaMode"));
                        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Price"));
                        PropertyChanged?.Invoke(this,
                                new PropertyChangedEventArgs("Instructions"));

                }
        }

        private bool _hot = false;
        public bool Hot {
                get => _hot;
                set {
                        _hot = value;
                        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Hot"));
                        PropertyChanged?.Invoke(this,
                                new PropertyChangedEventArgs("Instructions"));

                }
        }

        public PaleozoicPie(PieFilling filling) {
```

```
            Filling = filling;
        }

        public decimal Price {
            get {
                if (!ALaMode) return 3.00m;
                else return 4.00m;
            }
        }

        public string Description {
            get => Filling + " Palezoic Pie";
        }

        public List<string> Instructions {
            get {
                List<string> inst = new List<string>();
                if (Hot) inst.Add("Hot");
                if (ALaMode) inst.Add("A La Mode");

                return inst;
            }
        }
}
```

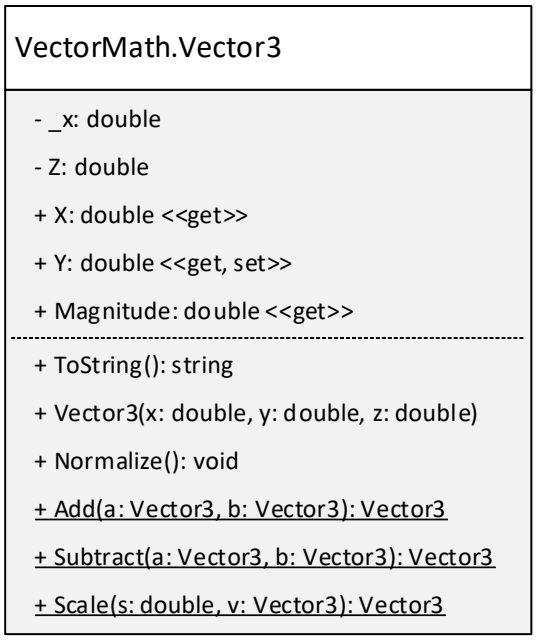
3. Complete the tests of *PaleozoicPie*:

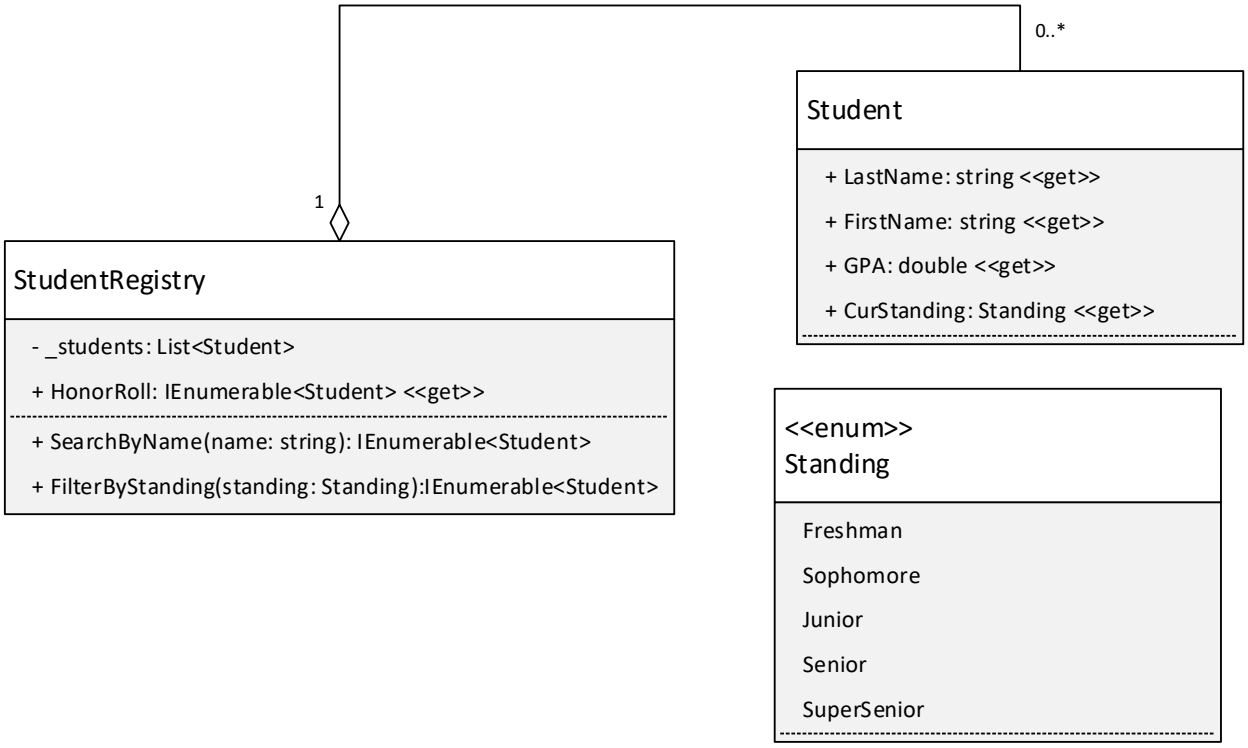
```
using Xunit;
public class PaleozoicPieTest
{
	[Fact]
	public void DefaultPriceShouldBeCorrect()
	{
		PaleozoicPie p = new(PieFilling.Cherry);

		Assert.Equal(3.00m, p.Price);

	}


	[Theory]
	[InlineData(PieFilling.Apple)]
	[InlineData(PieFilling.Blueberry)]
	[InlineData(PieFilling.Cherry)]
	[InlineData(PieFilling.Peach)]
	public void DescriptionShouldReflectFilling(PieFilling filling)
	{
		PaleozoicPie p = new(filling);
		Assert.Equal(filling + " Paleozoic Pie", p.Description);

	}

	//fill in the necessary information for the last test case here

	[Theory]
	[InlineData(PieFilling.Apple)]
	[InlineData(PieFilling.Blueberry)]
	[InlineData(PieFilling.Cherry)]
	[InlineData(PieFilling.Peach)]
	public void ChangingFillingInvokesPropertyChanged(PieFilling filling)
	{
		PaleozoicPie pie = new PaleozoicPie(PieFilling.Cherry);
		Assert.PropertyChanged("Filling", () => { pie.Filling = filling; });
	}
 }
```

4. Draw the UML diagram for the *Vector3* class below, including ALL fields, properties and methods.

```
namespace VectorMath
{
	public class Vector3
	{
		private double _x;
		public double X => _x;	//+ X: double <<get>>

		public double Y {get; set;}

		private double Z;

		public override string ToString()
		{
			return $"<{this.X}, {this.Y}, {this.Z}>";
		}
		public Vector3(double x, double y, double z)
		{
			_x = x;
			Y = y;
			Z = z;
		}

		public double Magnitude
		{
			get
			{
				return Math.Sqrt(Math.Pow(this.X,2) +
					Math.Pow(this.Y,2) +
					Math.Pow(this.Z,2))
			}
		}

		public void Normalize()
		{
			double magnitude = this.Magnitude();
			this._x /= magnitude;
			this.Y /= magnitude;
			this.Z /= magnitude;
		}
		  .                    .
//error! this should have had "a" as a param too
		public static Vector3 Add(Vector3 b) {
			return new Vector3(a.X + b.X, a.Y + b.Y, a.Z + b.Z);
		}

		public static Vector3 Subtract(Vector3 a, Vector3 b)
		{
```

```
                return new Vector3(a.X - b.X, a.Y - b.Y, a.Z + b.Z);
            }

            public static Vector3 Scale(double s, Vector3 v)
            {
                return new Vector3(s * v.X, s * v.Y, s * v.Z);
            }
        }
}
```





5. Consider the UML above. Assuming the private variable *_students* is already initialized, complete the *StudentRegistry* members below. Students with a GPA of 3.5 and above are on the Honor Roll. The search by name filter should find all students whose first or last names contain the parameter string. You should use LINQ queries to do your filters.

```
public IEnumerable<Student> HonorRoll
{
	get
	{
		return _students.Where(s => s.GPA >= 3.5);
		//return from s in _students where s.GPA >= 35 select s;
	}
}

public IEnumerable<Student> SearchByName(string name)
{
	return _students.Where(s => s.FirstName.Contains(name) ||
				s.LastName.Contains(name));

}

public IEnumerable<Student> FilterByStanding(Standing standing)
{
	return _students.Where(s => s.CurStanding == standing);

}
```