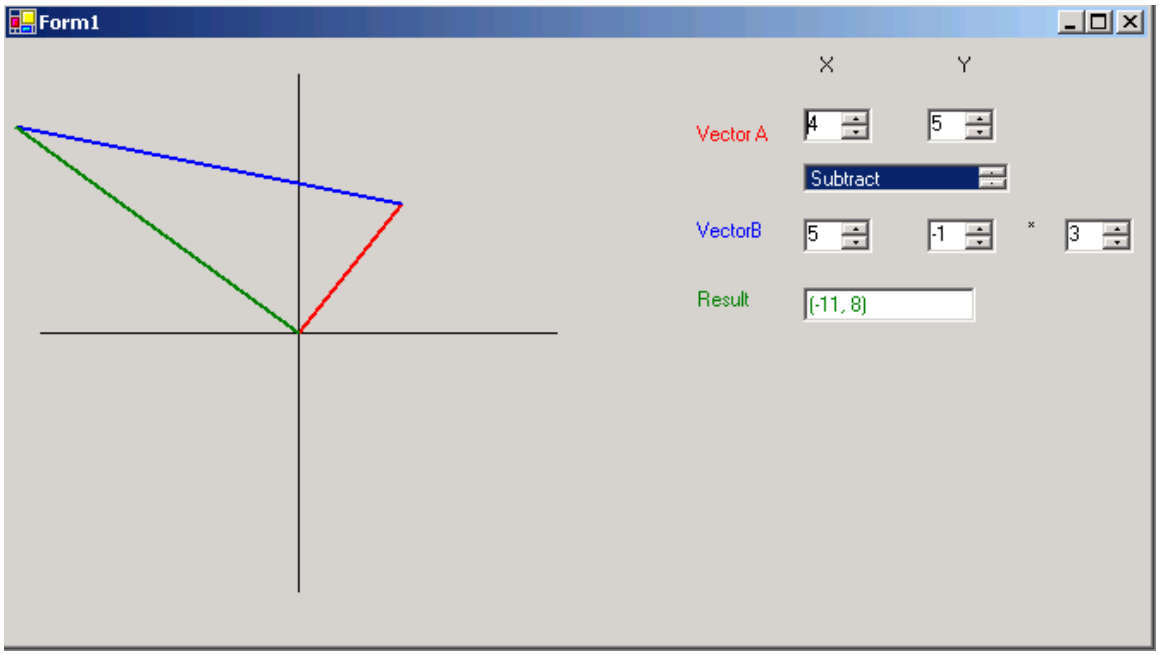
Operacije nad vektorima

Vaš zadatak je da kreirate aplikaciju koja će sabirati i oduzimati vektore i koja će rezultat prikazivati na formi. Korisnik treba da može nezavisno da menja vrednosti x i y komponenata vektora A i B. Vektor B može biti pomnožen skalarom. Na kraju, korisnik treba da može da odabere da li će sabrati ili oduzeti vektore. Na sledećoj slici prikazano je kako bi aplikacija trebalo da izgleda.



Implementacija klase Vector

Klasa Vector se sastoji od dva svojstava X i Y, nekoliko predefisinanih operatora ==, !=, +, - i *, kao i od nekoliko metoda ToString, Parse and GetHashCode. Klasa Vector neće sadržati metode za crtanje vektora – to će raditi korisnički interfejs koji će biti implementiran kroz formu.

Kreiranje projekta i klase Vector

Projekat VectorAlgebra sastoji se samo od forme i klase Vector.

1. Kreirajte novi projekat tipa Visual C# Windows Application i nazovite ga VectorAlgebra.
2. Dodajte u projekat novi fajl koji sadrži klasu Vector.
3. Dodajte sledeća polja i svojstva u klasu Vector

```
private int m_x;
public int X
{
        get {return m_x;}
        set {m_x = value;}
}
```

```
private int m_y;
public int Y
{
        get {return m_y;}
        set {m_y = value;}
}
```

4. Modifikujte konstruktor na sledeći način:

```
public Vector(int x, int y)
{
        m_x = x;
        m_y = y;
}
```

Predefinisanje operatora == i !=

Ako se operator == ne predefiniše vratiće vrednost boolean tipa koja će pokazati da li dve reference klase Vector ukazuju na isti objekat. Za našu upotrebu ovde ovo nije od nekog velikog značaja. Nama zapravo treba da utvrdimo da li vektori zapravo imaju isti smer i intenzitet. Zbog toga ćemo predefinisati operator == kako bi mu dali drugačije značenje.

Kod predefinisanja operatora ==, u jeziku C#, postoje određena pravila kojih se moramo držati. Najpre, ako se predefiniše operator ==, takođe se mora predefinisati i operator != . Po konvenciji jezika, kada se predefiniše operator == i metode Equals i GetHashCode takođe se moraju predefinisati. Takođe, po konvenciji jezika, operator == nikako ne bi smeo da generiše (baca) izuzetak, već, umesto toga treba da vraća vrednost false.

Dodajte sledeći kod za predefinisanje operatora ==:

```
public static bool operator ==(Vector aVector, Vector bVector)
{
        return (aVector.X == bVector.X)&&(aVector.Y == bVector.Y);
}
```

Sintaksa za predefinisanje operatora je takva da zahteva definisanje javne statičke metode koja vraća određeni tip podatka, a prihvata parametre onog tipa za koji hoćete da definišete operatorsku funkciju. Kasnije ćete videti, kod operatora *, da parametri ne moraju baš da budu istog tipa, ali najčešće jesu.Operator == je operator poređenja pa stoga vraća logičku (boolean) vrednost. A sada dodajte sledeći kod za predefinisanje operatora !=:

```
public static bool operator !=(Vector aVector, Vector bVector)
{
        return !(aVector == bVector);
}
```

Predefinisanje metofa Equals i GetHashCode

I metoda Equals kao i operator == koji dolaze iz System.Object klase, koja je bazna klasa svim klasama koje kreirate, vraćaju true ako dve reference ukazuju na istu instancu klase. Pošto ste predefinisali operator == i dali mu novo značenje sada bi trebalo predefinisati i metodu Equals i dati joj isto značenje kakvo ima i operator ==, a to je poređenje dva vektora po smeru i intenzitetu.

Dodajte sledeći kod da biste predefinisali Equals metodu:

```
public override bool Equals(object o)
{
	return (o is Vector) && (this == (Vector)o);
}
```

Pošto Equals metoda ima instancu klase Object kao parametar potrebno je da proverite da li je prosleđeni atribut odgovarajućeg tipa.

O svrsi metode GetHashCode pogledajte u helpu, a ovde će biti dat samo kod za njeno predefinisanje:

```
public override int GetHashCode()
{
	return this.X;
}
```

Predifinisanje unarnog – operatora

Dodajte sledeći kod za predefinisanje unarnog – operatora:

```
public static Vector operator -(Vector vector)
{
	return new Vector(-vector.X, -vector.Y);
}
```

Ovde moćete videti da je i on javna statička metoda, ali, naravno vraća vrednost tipa Vector. Atribut je takođe tipa Vector.

Predefinisanje binarnih + i – operatora

Sabiranje i oduzimanje dva vektora za posledicu ima nastajanje trećeg, novog, vektora. U matematici bi se ovo pisalo kao vektorC = vektorA + vektorB. Dakle, kada predefinišete operatore sabiranja i oduzimanja rezultat treba da bude novi vektor, a vrednosti učesnika u računskim operacijama treba da ostanu nepromenjene. Dodajte sledeći kod za + operator:

```
public static Vector operator +(Vector aVector,Vector bVector)
{
	return new Vector(aVector.X + bVector.X, aVector.Y + bVector.Y);
}
```

Dodajte sledeći kod za binarni – operator:

```
public static Vector operator -(Vector aVector,Vector bVector)
{
	return aVector + (-bVector);
}
```

Obratite pažnju da je za realizaciju oduzimanja upotrebljen unarni – operaor.

Predefinisanje operatora * u svrhu skalarnog množenja

Do sada definisani operatori koriste jedino parametre koji su instance klase Vector. Takođe se mogu definisati i operatori koji imaju različite tipove operanada tako što se promene parametri predefinisanih metoda. Dodajte sledeći kod za predefinisanje operatora * u svrhu množenja vektora skalarom:

```
public static Vector operator *(int scalar,Vector vector)
{
	return new Vector(scalar *vector.X, scalar * vector.Y);
}
```

Kada se koristi ovaj operator on mora biti pozvan na sledeći način: vektorB = 2 * vektorA. Ako biste hteli da ovu vrstu množenja izvedete kao vektorB = vektorA * 2, onda biste morali da definišete još jedan operator * u kome bi parametri bili u obrnutom redosledu:

```
public static Vector operator *(Vector vector, int scalar)
{
	return new Vector(scalar *vector.X, scalar * vector.Y);
}
```

Definisanje metoda ToString i Parse

Metoda ToString služi za prikazivanje vrednosti nekog objekta u vidu stringa. Ako se ne predefiniše njen rezulatat biće naziv klase za čiju instancu je pozvana. Za predefinisanje metode ToString dodajte sledeći kod:

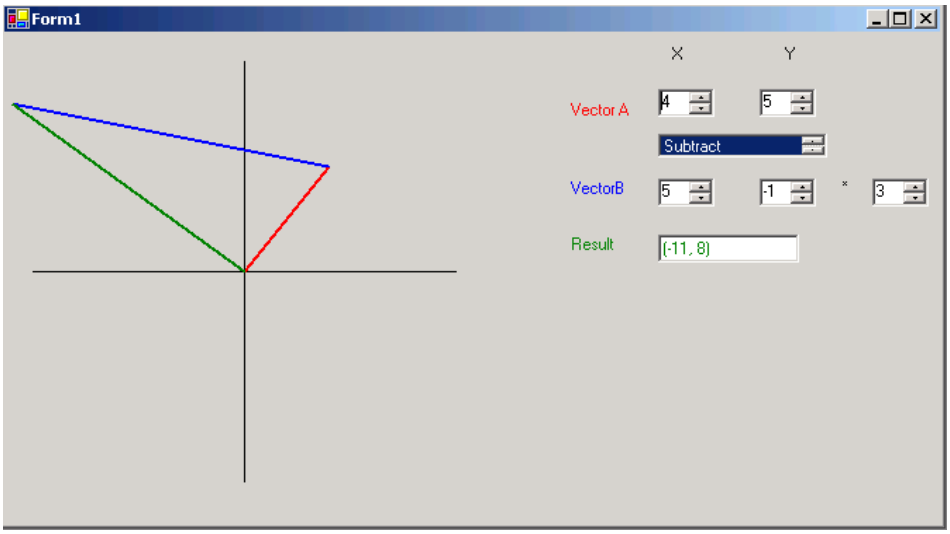
```
public override string ToString()
{
	return string.Format("({0}, {1})", m_x, m_y);

}
```

Metoda Parse treba da parsira ulaz kako bi prihvatila samo cele brojeve za vrednosti komponenti vektora. Ona očekuje unos tipa 7, 5 ili -3, 2 kao validan. U suprotnom generiše izuzetak. Za njenu implementaciju dodajte sledeći kod:

```
public static Vector Parse(string vectorString)
{
	try
	{
		string[] values = vectorString.Split("(, )".ToCharArray());
		int x = int.Parse(values[1]);
		int y = int.Parse(values[3]);
		return new Vector(x,y);
	}
	catch
	{
		throw new ArgumentException("Unable to parse `" + vectorString
					+"' into a Vector instance.");
	}
}
```

Implementacija aplikacije VectorAlgebra

Korisnički interfejs koji će omogućiti korisniku da kreira dva vektora i da vrši operacije nad njima treba da bude kao na sledećoj slici. Drugi vektor treba da ima mogućnost množenja skalarom.



Dodavanje elemenata korisničkog interfejsa

Da bi korisnik mogao da podešava vrednosti vektora A treba da dodate sledeće komponente i da njihova svojstva podesi po sledećoj tabeli:

Kontrola	Svojstvo	Vrednost
Label	Text	X
Label	Text	Y
Label	Text ForeColor	Vektor A Red
NumericUpDown	Name	XVectorA
NumericUpDown	Name	YVectorA

Dodajte kontrolu ListBox koja će biti korišćena za određivanje operacije koja će se vršiti nad vektorima. Postavite vrednost njenog svojstva Name na functions.

Da bi korisnik mogao da podešava vrednosti vektora B treba da dodate sledeće komponente i da njihova svojstva podesi po sledećoj tabeli:

Kontrola	Svojstvo	Vrednost
Label	Text	X
Label	Text	Y

Label	Text ForeColor	Vektor B Blue
NumericUpDown	Name	XVectorB
NumericUpDown	Name	YVectorB
Label	Text	*
NumericUpDown	Name Minimum Maximum Value Increment DecimalPlaces	Scalar -3 3 1 1 0

Dodajte kontole koje će predstaviti rezultat operacije:

Da bi korisnik mogao da podešava vrednosti vektora A treba da dodate sledeće komponente i da njihova svojstva podesi po sledećoj tabeli:

Kontrola	**Svojstvo**	**Vrednost**
Label	Text ForeColor	Result Green
TextBox	Text ForeColor Name	 Green Result

Sada selektujte sve četiri NumericUpDown kontrole za podešavanje komponenti vektora i postavite im svojsva na vrednosti po sledećoj tabeli:

Da bi korisnik mogao da podešava vrednosti vektora A treba da dodate sledeće komponente i da njihova svojstva podesi po sledećoj tabeli:

Svojstvo	**Vrednost**
Minimum	-5
Maximum	5
Value	0
Increment	1
DecimalPlaces	0

Dodavanje metoda za crtanje

Sada dodajte metode koje će iscrtavativektore na formu. Graf će biti predstavljen koordinatnim osama na kojima su označene vrednosti od -10 do 10, Graf će biti nacrtan na formi između piksela 20 i 170. Stoga, jedinično rastojanje na grafu biće 15 piksela. Graf je odmaknut za po 20 piksela od vrha i leve strane forme.

Kreirajte metodu za obradu događaja Paint tako što ćete kliknuti na dugme Events panela Properties nakon što odaberete formu. Dvostruki klik na na metodu Paint odvešće vas do mesta gde treba da ukucate kod za obradu ovog događaja. Ovaj kod crta ose na grafu:

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{
       e.Graphics.DrawLine(Pens.Black, 20, 170, 320, 170);
       e.Graphics.DrawLine(Pens.Black, 170, 20, 170, 320);
}
```

Sada dodajte u klasu sledeću metodu koja preslikava poziciju u grafu (-10 do 10) u pozicije na formi (20 do 170):

```
private Point VectorToPoint(Vector vector)
{
       return new Point(vector.X*15 +170,-vector.Y*15 +170);
}
```

Dodajte sada sledeće dve metode za crtanje vektora na formi. Prva metoda crta vektor iz koordinatnog početka, a druga iz kraja drugog vektora i koristi se da crta vektore prilikom sabiranja:

```
private void DrawVector(Vector vector,Color color)
{
       Point origin =VectorToPoint(new Vector(0,0));
       Point end =VectorToPoint(vector);
       this.CreateGraphics().DrawLine(
              new Pen(new SolidBrush(color),2),origin,end);
}
private void DrawVector(Vector aVector,Vector bVector,Color color)
{
       Point origin =VectorToPoint(bVector);
       Point end =VectorToPoint(aVector +bVector);
       this.CreateGraphics().DrawLine(
              new Pen(new SolidBrush(color),2),origin,end);
}
```

Dodavanje kontrolne logike

Ovde ćete iskoristiti delegatske funkcije da biste pozvali operacije sabiranja i oduzimanja vektora.

Dodajte sledeću delegatsku funkciju po kojoj ćete kasnije realizovati operacije sabiranja, oduzimanja i poređenja:

```
private delegate void VectorMath(Vector a,Vector b);
```

Dodajte i jednu instancu klase SortedList koja će poslužiti da popuni podacima malopre dodatu kontrolu tipa ListBox:

```
System.Collections.SortedList m_maths = new System.Collections.SortedList();
```

Dodajte još i privatne propertije koji će konvertovati vrednosti kontrola NumericUpDown u instance klase Vector:

```
private Vector VectorA
{
	get
	{
		return new Vector((int)this.XVectorA.Value,
			(int)this.YVectorA.Value);
	}
}
private Vector VectorB
{
	get
	{
		return new Vector((int)this.XVectorB.Value,
			(int)this.YVectorB.Value);
	}
}
```

Dodajte metode za sabiranje, oduzimanje i poređenje vektora (sve su napravljene po delegatskom šablonu):

```
private void AddVectors(Vector a,Vector b)
{
	DrawVector(a,Color.Red);
	DrawVector(b,a,Color.Blue);
	Vector sum =a +b;
	DrawVector(sum,Color.Green);
	this.result.Text =sum.ToString();
}
private void SubtractVectors(Vector a,Vector b)
{
	DrawVector(a,Color.Red);
	DrawVector(-b,a,Color.Blue);
	Vector difference =a -b;
	DrawVector(difference,Color.Green);
	this.result.Text =difference.ToString();
}
private void AreEqual(Vector a,Vector b)
{
	bool equal =(a ==b);
	this.result.Text =equal.ToString();
}
```

Kreirajte metodu za obradu forminog događaja Load, u kojoj će se inicijalizovati vrednostima ListBox sa operacijama. Inicijalizacija je urađena korišćenjem instance klase SortedList (ovo se moglo uraditi i upisivanjem vrednosti za svojstvo Items instance klase ListBox):

```
private void Form1_Load(object sender, System.EventArgs e)
{
	m_maths.Add("Add", new VectorMath(AddVectors));
	m_maths.Add("Subtract", new VectorMath(SubtractVectors));
	m_maths.Add("Are equal", new VectorMath(AreEqual));
	functions.DataSource = m_maths.Keys;

}
```

Povezivanje korisičkog interfejsa sa događajima

Izgled vektora na grafu bi trebalo da se menja svaki put kada korisnik promeni vrednost u nekom od komponenata tipa NumericTextBox ili izabere drugu operaciju u ListBox-u. Ovo znači da će implementacija za ValueChanged događaje svih NumericUpDown komponenti i SelectedIndexChanged metode ListBox-a biti ista, odnosno da će sve one zapravo koristiti isti kod. Ovo se dakle može iskoristiti da se svim ovom metodama za obradu događaja (event handlers) pridruži isti metod. Tako, dodajte sledeću metodu u klasu:

```
private void VectorChanged(object sender,System.EventArgs e)
{
       this.Refresh();
       VectorMath theMath = (VectorMath)m_maths[functions.Text];
       theMath(this.VectorA, (int)scalar.Value * this.VectorB);
}
```

Ova metoda pribavlja aktuelni tekst iz ListBox-a, i na osnovu njega kreira odgovarajući delegatski objekat. U ListBox-u su upisane vrednosti ekvivalentne nazivima metoda napravljenih po delegatskom modelu. Nakon toga se poziva odgovarajuća funkcija čiji je delegatski objekat kreiran.

Na kraju, postavite da ova metoda obrađuje ValueChanged događaj svih NumericUpDown komponenti. Takođe postavite da ona obrađuje i događaj SelectedIndexChanged ListBox-a functions.