E – Expressions and Math

Code Samples – Documentation

# Examples Overview

The following examples are used to illustrate this topic.

1. **Calculator** – This class introduces simple arithmetic operations, demonstrating simple addition and multiplication.
2. **Person** - This adaptation of the person class changes the previous design by replacing the age field with a "read-only" field for the person's birth date. The approximate age of the person is then calculated based on the current date and the birth date.
3. **Account** - This class illustrates simple addition and calculation by allowing deposits and withdrawals. Note that changes to the balance can now only be made through deposits and withdrawals; the balance is now "read-only".
4. **ElapsedTime** – This class demonstrates overloaded constructors and introduces the concepts of operator precedence and integer division.
5. **ResolveExpressions** – This class is used in conjuncture with several sample expressions that illustrate operator precedence and automatic type conversion.
6. **Circle** - This class represents a simple circle of a specified diameter. The radius, area and circumference are calculated.
7. **Square** - This class represents a simple square with a specified length for its side. The area and perimeter are calculated.
8. **Fraction** - This class represents a fraction as a numerator and denominator. It provides the double equivalent of the fraction's value as well as a string representation that uses the numerator and denominator. It demonstrates type casting and the integer division issue.
9. **Angle** - This class represents an angle and provides the value in the following units: degrees, radians and grads. It also gives a simple example of unicode characters (for degrees).
10. **StockItem** - This class represents an item that is part of an inventory. The item has an item name, a cost and a profit margin (which can be positive or negative). By using the profit margin, it can derive the price of the item. This example illustrates rounding.
11. **Die** - This class represents a single six-sided die. This example is used to illustrate random number generation and casting.
12. **ParkingCounter** - This class represents a simple counter to monitor whether a parking lot is full or not; it tracks vehicles entering and leaving the parking lot and allows the counter to be reset when the lot is full or empty. This class illustrates increment and decrement operators and/or the assignment increment or assignment decrement operators.
13. **QuadradicEquation** - This class is used to solve for the two possible values of a quadradic formula where quadradic equals zero. It is based off of the following formula.  
    x = \frac{-b \pm \sqrt {b^2-4ac}}{2a},  
    This sample illustrates order of operations and parentheses.

# Calculator

This class introduces simple arithmetic operations, demonstrating simple addition and multiplication.

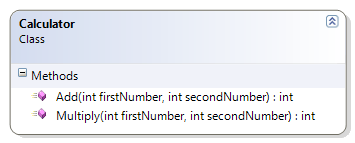
## Problem Statement

Write the code that will act as a calculator for doing math. This first version must be a working prototype; as a prototype, it does not have to support all of the features of the final product.

The solution must meet the following requirements:

* Should add two whole numbers.
* Should multiply two whole numbers.

Use the following class diagram when creating your solution. Since this class does not have properties or fields, make the methods ***static***.



## Code Solution

public class Calculator

{

public static int Add(int firstNumber, int secondNumber)

{

return firstNumber + secondNumber;

}

public static int Multiply(int firstNumber, int secondNumber)

{

return firstNumber \* secondNumber;

}

}

# Person

This adaptation of the person class changes the previous design by replacing the age field with a "read-only" field for the person's birth date. The approximate age of the person is then calculated based on the current date and the birth date.

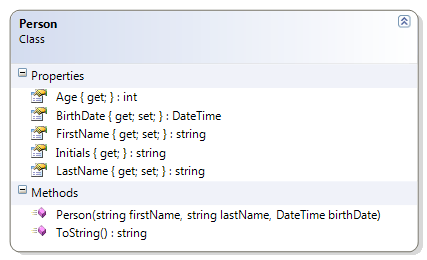
## Problem Statement

Write the code that will represent a person with a first and last name and a date of birth.

The solution must meet the following requirements (new requirements are in ***green, bold italic*** font):

* Should get and set the first and last name
* ***Should get the birth date***
* ***Should get the person’s approximate age (which is the age that the person will turn to in the current year)***
* ***Should get the person’s initials***
* Should override ToString() to get the person’s full name (as first name then last name)

Use the following class diagram when creating your solution.



## Code Solution

public class Person

{

public string FirstName { get; set; }

public string LastName { get; set; }

public DateTime BirthDate { get; private set; }

public int Age

{

get

{

int currentAge = 0;

currentAge = DateTime.Today.Year - BirthDate.Year;

return currentAge;

}

}

public Person(string firstName, string lastName, DateTime birthDate)

{

FirstName = firstName;

LastName = lastName;

BirthDate = birthDate;

}

public override string ToString()

{

return FirstName + " " + LastName;

}

public string Initials

{

get

{

return FirstName[0] + "." + LastName[0] + ".";

}

}

}

# Account

This class illustrates simple addition and calculation by allowing deposits and withdrawals. Note that changes to the balance can now only be made through deposits and withdrawals; the balance is now "read-only".

## Problem Statement

Write the code that will represent a simple bank account.

The solution must meet the following requirements (new requirements are in ***green, bold italic*** font):

* Should get the bank name, branch number, institution number, account number, balance, overdraft limit, and account type
* Should allow the overdraft limit to be set
* ***Should support deposits and withdrawals***

Use the following class diagram when creating your solution.



## Code Solution

public void Withdraw(double amount)

{

Balance -= amount;

}

public void Deposit(double amount)

{

Balance += amount;

}

# ElapsedTime

This class demonstrates overloaded constructors and introduces the concepts of operator precedence and integer division.

## Problem Statement

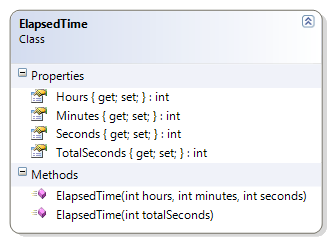
Write the code that will represent a period of elapsed time for a competitor in a marathon. It should be able to represent its information in two forms:

* Hours, minutes and seconds, and
* Total seconds.

The solution must meet the following requirements:

* Should calculate the hours, minutes and seconds given the total seconds
* Should calculate the total seconds given the hours, minutes and seconds

Use the following class diagram when creating your solution.



## Code Solution

public class ElapsedTime

{

public ElapsedTime(int hours, int minutes, int seconds)

{

TotalSeconds = hours \* 60 \* 60;

TotalSeconds += minutes \* 60;

TotalSeconds += seconds;

Hours = hours;

Minutes = minutes;

Seconds = seconds;

}

public ElapsedTime(int totalSeconds)

{

Hours = totalSeconds / (60 \* 60);

Minutes = (totalSeconds - Hours \* 60 \* 60) / 60;

Seconds = totalSeconds - Hours \* 60 \* 60 - Minutes \* 60;

TotalSeconds = totalSeconds;

}

public int Hours { get; private set; }

public int Minutes { get; private set; }

public int Seconds { get; private set; }

public int TotalSeconds { get; private set; }

}

# ResolveExpressions

This class is used in conjuncture with several sample expressions that illustrate operator precedence and automatic type conversion.

## Problem Statement

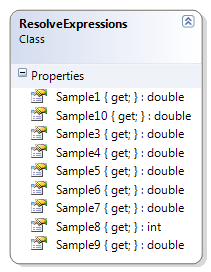
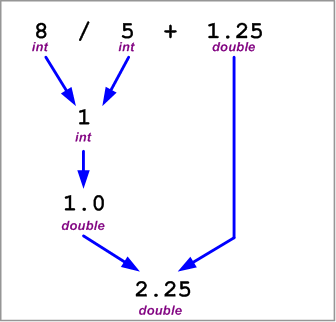
Write the code that will provide the final value for the expressions in the following exercise (as a solution for a student’s exercise).

### Expressions Exercise

On a piece of paper, evaluate the following expressions to show the final value and the data type of the final value. Show the order in which the operations are evaluated.

1. 10.0 + 15 / 2 + 4.3
2. 8 / 5 + 1.25
3. 10.0 + 15.0 / 2 + 4.3
4. 3.0 \* 4 / 6 + 6
5. 3.0 \* (4 % 6) + 6
6. 3 \* 4.0 / 6 + 6
7. 20.0 - 2 / 6 + 3
8. 10 + 17 % 3 + 4
9. (10 + 17) % 3 +4.0
10. 10 + 17 / 4.0 + 4

The solution for question 2 is provided as an example. Use the accompanying class diagram when creating your coded solution as proof of your final answers.



## Code Solution

namespace Topic.E.Examples

{

public class ResolveExpressions

{

public static double Sample1

{

get

{

return 10.0 + 15 / 2 + 4.3;

}

}

public static double Sample3

{

get

{

return 10.0 + 15.0 / 2 + 4.3;

}

}

public static double Sample4

{

get

{

return 3.0 \* 4 / 6 + 6;

}

}

public static double Sample5

{

get

{

return 3.0 \* (4 % 6) + 6;

}

}

public static double Sample6

{

get

{

return 3 \* 4.0 / 6 + 6;

}

}

public static double Sample7

{

get

{

return 20.0 - 2 / 6 + 3;

}

}

public static int Sample8

{

get

{

return 10 + 17 % 3 + 4;

}

}

public static double Sample9

{

get

{

return (10 + 17) % 3 + 4.0;

}

}

public static double Sample10

{

get

{

return 10 + 17 / 4.0 + 4;

}

}

}

}

# Circle

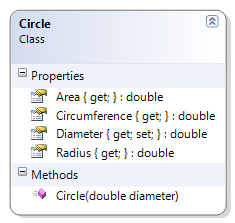
This class represents a simple circle of a specified diameter. The radius, area and circumference are calculated.

## Problem Statement

Write the code for the Circle class. The solution must meet the following requirements:

* Should get and set the diameter
* Should calculate the area, radius, and circumference

Use the following class diagram when creating your solution.



## Code Solution

using System;

namespace Topic.E.Examples

{

public class Circle

{

public Circle(double diameter)

{

this.Diameter = diameter;

}

public double Diameter { get; set; }

public double Radius

{

get

{

return Diameter / 2;

}

}

public double Circumference

{

get

{

return Math.PI \* Diameter;

}

}

public double Area

{

get

{

return Math.PI \* Radius \* Radius;

}

}

}

}

# Square

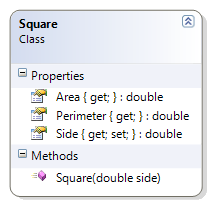
This class represents a simple square with a specified length for its side. The area and perimeter are calculated.

## Problem Statement

Write the code for the Square class. The solution must meet the following requirements:

* Should get and set the length of the side of the square
* Should calculate the area and perimeter

Use the following class diagram when creating your solution.



## Code Solution

namespace Topic.E.Examples

{

public class Square

{

public Square(double side)

{

this.Side = side;

}

public double Side { get; set; }

public double Area

{

get { return Side \* Side; }

}

public double Perimeter

{

get { return Side \* 4; }

}

}

}

# Fraction

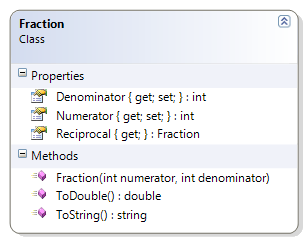
This class represents a fraction as a numerator and denominator. It provides the double equivalent of the fraction's value as well as a string representation that uses the numerator and denominator. It demonstrates type casting and the integer division issue.

## Problem Statement

Write the code for the Fraction class. The solution must meet the following requirements:

* Should get the string representation of the fraction, as “numerator/denominator”
* Should get the numeric value of the fraction (as a real number)
* Should get the reciprocal of the fraction

Use the following class diagram when creating your solution.



## Code Solution

public class Fraction

{

public int Numerator { get; private set; }

public int Denominator { get; private set; }

public Fraction(int numerator, int denominator)

{

Numerator = numerator;

Denominator = denominator;

}

public Fraction Reciprocal

{

get { return new Fraction(Denominator, Numerator); }

}

public override string ToString()

{

string stringValue = "";

stringValue += Numerator + "/" + Denominator;

return stringValue;

}

public double ToDouble()

{

// The casting of numerator to a double helps

// ensure that we don't lose any fractional

// portion due to integer division.

double value = (double)(Numerator) / Denominator;

return value;

}

}

# Angle

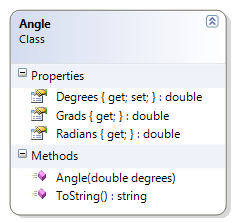
This class represents an angle and provides the value in the following units: degrees, radians and grads. It also gives a simple example of unicode characters (for degrees).

## Problem Statement

Write the code for the Angle class. The solution must meet the following requirements:

* Should get and set the angle’s value (in degrees)
* Should calculate the equivalent angle in Radians and Grads, using the following formulas:
  + Radians = Degrees \* (π / 180)
  + Grads = Radians \* (200 / π)
* Should override the ToString() method to return the angle in degrees, in the following format:
  + degrees°
  + The Unicode character for the degrees symbol (°) is '\u00B0'

Use the following class diagram when creating your solution.



## Code Solution

using System;

namespace Topic.E.Examples

{

public class Angle

{

public Angle(double degrees)

{

this.Degrees = degrees;

}

public double Degrees { get; set; }

public double Radians

{

get

{

double radians = Degrees \* (Math.PI / 180);

return radians;

}

}

public double Grads

{

get

{

double grads = Radians \* (200 / Math.PI);

return grads;

}

}

public override string ToString()

{

// http://unicode.org/notes/tn28/UTN28-PlainTextMath.pdf

// Page 40 of the above reference

return Degrees.ToString() + '\u00B0';

}

}

}

# StockItem

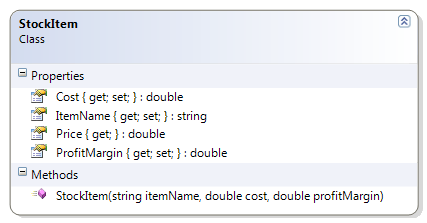
This class represents an item that is part of an inventory. The item has an item name, a cost and a profit margin (which can be positive or negative). By using the profit margin, it can derive the price of the item. This example illustrates rounding.

## Problem Statement

Write the code for the StockItem class. The solution must meet the following requirements:

* Should get and set the name, cost and profit margin of the stock item
* Should represent the profit margin as a percent; a value of 45 means 45%
* Should calculate the price of the item, to the nearest cent
  + Use the rounding where values under a half-cent are rounded down and values greater than or equal to a half-cent are rounded up

Use the following class diagram when creating your solution.



## Code Solution

using System;

namespace Topic.E.Examples

{

public class StockItem

{

public double Cost { get; set; }

public double ProfitMargin { get; set; }

public StockItem(string itemName, double cost, double profitMargin)

{

this.ItemName = itemName;

this.Cost = cost;

this.ProfitMargin = profitMargin;

}

public string ItemName { get; set; }

public double Price

{

get

{

// Round to the nearest cent

double price = Cost;

price += Cost \* (ProfitMargin / 100);

return Math.Round(price \* 100) / 100.0;

}

}

}

}

# Die

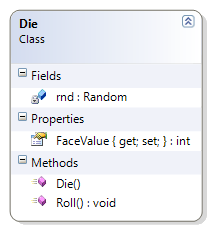
This class represents a single six-sided die. This example is used to illustrate random number generation and casting.

## Problem Statement

Write the code for the Die class. The solution must meet the following requirements:

* Should generate a random value from 1 to 6, when initially created and when re-rolled
* Should get the face value of the die

Use the following class diagram when creating your solution. Note that this uses the Random class as a private static field.



## Code Solution

using System;

namespace Topic.E.Examples

{

public class Die

{

private static Random rnd = new Random();

public Die()

{

Roll();

}

public int FaceValue { get; private set; }

public void Roll()

{

FaceValue = rnd.Next(6000) % 6 + 1;

}

}

}

# ParkingCounter

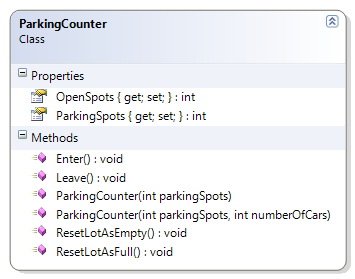
This class represents a simple counter to monitor whether a parking lot is full or not; it tracks vehicles entering and leaving the parking lot and allows the counter to be reset when the lot is full or empty. This class illustrates increment and decrement operators and/or the assignment increment or assignment decrement operators.

## Problem Statement

Write the code that will monitor vehicles entering and leaving a parking lot. The solution must meet the following requirements:

* Should track vehicles entering
* Should track vehicles leaving
* Should get total parking spots
* Should get open (empty) spots
* Should reset lot as full (that is, fill the parking lot)
* Should reset lot as empty (that is, clear all the parking spots of vehicles)

Use the following class diagram when creating your solution.



## Code Solution

namespace Topic.E.Examples

{

public class ParkingCounter

{

public int ParkingSpots { get; private set; }

public int OpenSpots { get; private set; }

public ParkingCounter(int parkingSpots)

{

this.ParkingSpots = parkingSpots;

this.OpenSpots = parkingSpots;

}

public ParkingCounter(int parkingSpots, int numberOfCars)

{

this.ParkingSpots = parkingSpots;

this.OpenSpots = this.ParkingSpots - numberOfCars;

}

public void Leave()

{

OpenSpots++;

}

public void Enter()

{

OpenSpots--;

}

public void ResetLotAsEmpty()

{

OpenSpots = ParkingSpots;

}

public void ResetLotAsFull()

{

OpenSpots = 0;

}

}

}

# QuadradicEquation

This class is used to solve for the two possible values of a quadratic formula where quadratic equals zero. It is based off of the following formula.  
x = \frac{-b \pm \sqrt {b^2-4ac}}{2a},  
This sample illustrates order of operations and parentheses.

## Problem Statement

Write the code that will represent a quadratic equation that has a higher and lower root. It is to use the Quadratic formula, which states:

For ***ax2 + bx + c = 0***, the value of ***x*** is given by

x = \frac{-b \pm \sqrt {b^2-4ac}}{2a},

More information on the quadratic formula can be found at <http://www.purplemath.com/modules/quadform.htm>.

The solution must meet the following requirements:

* Should get the lower root, using the formula
* Should get the higher root, using the formula
* Should overload the ToString() method to represent the quadratic formula showing the values for ***a***, ***b*** and ***c*** in the following format:  
  ***ax2 + bx + c = 0***  
  For example, given the values of 1, 3 and -4 for ***a***, ***b*** and ***c*** respectively, the method should produce  
  ***1x2 + 3x + -4 = 0***

Use the accompanying class diagram when creating your solution.

## Code Solution

using System;

namespace Topic.E.Examples

{

public class QuadraticEquation

{

private int a;

private int b;

private int c;

public QuadraticEquation(int a, int b, int c)

{

this.a = a;

this.b = b;

this.c = c;

}

public double LowerRoot

{

get

{

double value;

value = (-b - Math.Sqrt(b \* b - 4 \* a \* c)) / (2 \* a);

return value;

}

}

public double HigherRoot

{

get

{

double value;

value = (-b + Math.Sqrt(b \* b - 4 \* a \* c)) / (2 \* a);

return value;

}

}

public override string ToString()

{

return a.ToString() + "x^2 + " + b + "x + " + c + " = 0";

}

}

}