**Microsoft .NET Basics**

**Introduction to .NET**

.NET is a free, cross-platform, open-source developer platform for building many different types of applications. With .NET, you can use multiple languages, editors, and libraries to build for web, mobile, desktop, gaming, and IoT.

**CommentsinC#**

Comments are used for explaining code. Compiler ignores the comment entries. The

multiline comments in C# programs start with /\* and terminates with the characters

\*/ as shown below:

/\* This program demonstrates

The basic syntax of C# programming

Language \*/

Single-line comments are indicated by the '//' symbol. For example,

}//end class Rectangle

**Variables**

Variables are attributes or data members of a class, used for storing data. In the

preceding program, the *Rectangle* class has two

member variables

named *length* and *width*.

**Functions**

Functions are set of statements that perform a specific task. The member functions

of a class are declared within the class. Our sample class Rectangle contains three

member functions: *AcceptDetails*, *GetArea* and *Display*.

**Instantiating a Class**

In the preceding program, the class *ExecuteRectangle* contains the *Main()* method

and instantiates the *Rectangle* class.

**Identifiers**

An identifier is a name used to identify a class, variable, function, or any other user

defined item. The basic rules for naming classes in C# are as follows:

 A name must begin with a letter that could be followed by a sequence of letters,

digits (0 - 9) or underscore. The first character in an identifier cannot be a

digit.

 It must not contain any embedded space or symbol such as ? - +! @ # % ^ &

\* ( ) [ ] { } . ; : " ' / and \. However, an underscore ( \_ ) can be used.

 It should not be a C# keyword.

**ValueType**

Value type variables can be assigned a value directly. They are derived from the class

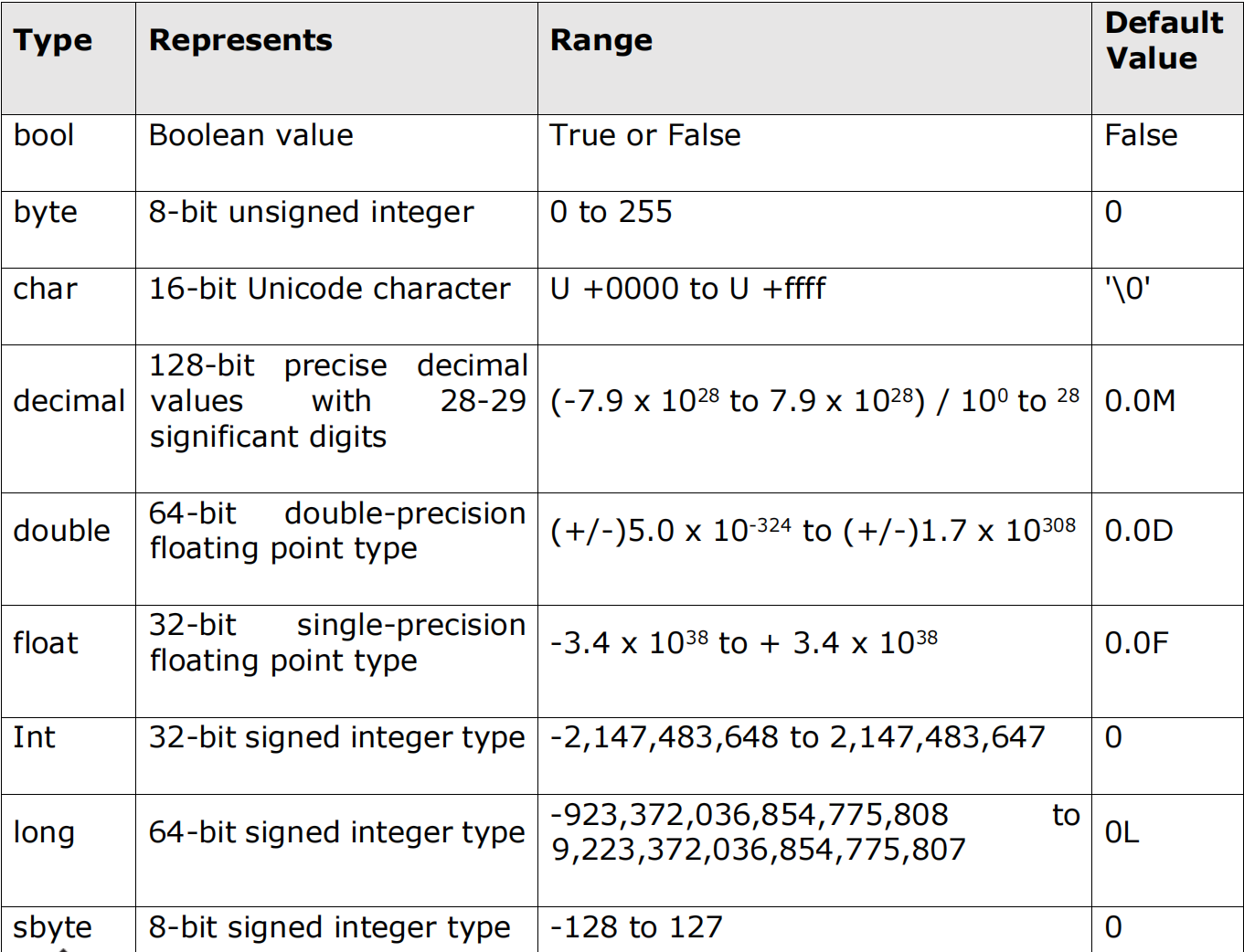
**System.ValueType**.

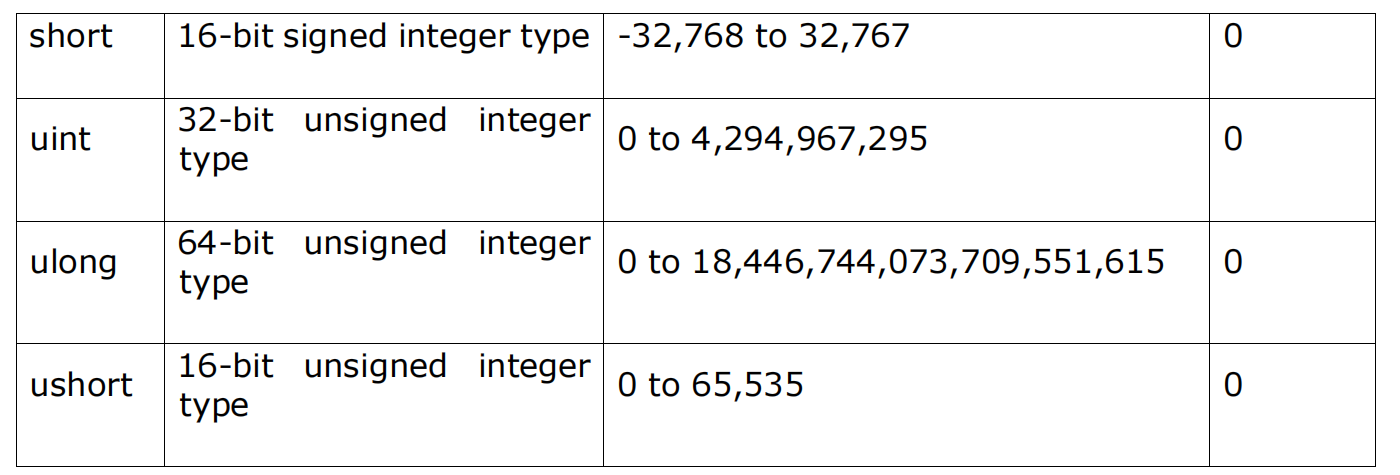
The value types directly contain data. Some examples are **int, char, and float**, which

stores numbers, alphabets, and floating point numbers, respectively. When you

declare an **int** type, the system allocates memory to store the value.

The following table lists the available value types in C# 2010:





To get the exact size of a type or a variable on a particular platform, you can use

the **sizeof** method. The expression *sizeof(type)* yields the storage size of the object

or type in bytes. Following is an example to get the size of *int* type on any machine:

**Eg**.

Data Type

namespace DataTypeApplication

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Size of int: {0}", sizeof(int));

Console.ReadLine();

}

}

}

When the above code is compiled and executed, it produces the following result:

Size of int: 4

**ReferenceType**

The reference types do not contain the actual data stored in a variable, but they

contain a reference to the variables.

In other words, they refer to a memory location. Using multiple variables, the

reference types can refer to a memory location. If the data in the memory location

is changed by one of the variables, the other variable automatically reflects this

change in value.

Example of **built-in** reference types are: **object**, **dynamic,** and **string**.

**ObjectType**

The **Object Type** is the ultimate base class for all data types in C# Common Type

System (CTS). Object is an alias for System.Object class. The object types can be

assigned values of any other types, value types, reference types, predefined or user

defined types. However, before assigning values, it needs type conversion.

When a value type is converted to object type, it is called **boxing** and on the other

hand, when an object type is converted to a value type, it is called **unboxing**.

object obj;

obj = 100; // this is boxing

**DynamicType**

You can store any type of value in the dynamic data type variable. Type checking for

these types of variables takes place at run-time.

Syntax for declaring a dynamic type is:

dynamic <variable\_name> = value;

For example,

dynamic d = 20;

Dynamic types are similar to object types except that type checking for object type

variables takes place at compile time, whereas that for the dynamic type variables

takes place at run time.

**StringType**

The **String Type** allows you to assign any string values to a variable. The string type

is an alias for the System.String class. It is derived from object type. The value for a

string type can be assigned using string literals in two forms: quoted and @quoted.

For example,

String str = "Tutorials Point";

19C#

A @quoted string literal looks as follows:

@"Tutorials Point";

The user-defined reference types are: class, interface, or delegate. We will

discuss these types in later.

**PointerType**

Pointer type variables store the memory address of another type. Pointers in

C# have the same capabilities as the pointers in C or C++.

Syntax for declaring a pointer type is:

type\* identifier;

For example,

char\* cptr;

int\* iptr;

We will discuss pointer types in the 'Unsafe Codes'.

**C# Basics**

**Variables and Data Types**

**Hello World**

using System;

namespace HelloWorldApplication

{

class HelloWorld

{

static void Main(string[] args)

{

/\* my first program in C# \*/

Console.WriteLine("Hello World");

Console.ReadKey();

}

}

}

**Rectangle Length, Width.**

using System;

namespace RectangleApplication

{

class Rectangle

{

// member variables

double length;

double width;

public void Acceptdetails()

{

length = 4.5;

width = 3.5;

}

public double GetArea()

{

return length \* width;

}

public void Display()

{

Console.WriteLine("Length: {0}", length);

Console.WriteLine("Width: {0}", width);

Console.WriteLine("Area: {0}", GetArea());

}

}

class ExecuteRectangle

{

static void Main(string[] args)

{

Rectangle r = new Rectangle();

r.Acceptdetails();

r.Display();

Console.ReadLine();

}

}

}

int age = 25;

string name = "Cdac"; bool isStudent = true;

**Control Flow Statements**

if (age > 18) {

Console.WriteLine("Adult"); }

else {

Console.WriteLine("Not an adult");

}

for (int i = 0; i < 5; i++) {

Console.WriteLine(i);

}

**Methods**

public void Greet() {

Console.WriteLine("Hello, World!"); }

public int Add(int a, int b) {

return a + b; }

**Integer Types – Example**

Consider an example in which we declare several variables of the integer

types we know, we initialize them and print their values to the console:

// Declare some variables

byte centuries = 20;

ushort years = 2000;

uint days = 730480;

ulong hours = 17531520;

// Print the result on the console

Console.WriteLine(centuries + " centuries are " + years +

" years, or " + days + " days, or " + hours + " hours.");

// Console output:

// 20 centuries are 2000 years, or 730480 days, or 17531520

// hours.

ulong maxIntValue = UInt64.MaxValue;

Console.WriteLine(maxIntValue); // 18446744073709551615

**Real Floating-Point Types**

**Special Values of the Real Types**

The real data types have also several special values that are not real numbers

but are mathematical abstractions:

- **Negative infinity -∞ (Single.NegativeInfinity)**. It is obtained

when for instance we are dividing **-1.0f** by **0.0f**.

- **Positive infinity +∞ (Single.PositiveInfinity)**. It is obtained

when for instance we are dividing **1.0f** by **0.0f**.

- **Uncertainty (Single.NaN)** – means that an invalid operation is

performed on real numbers. It is obtained when for example we divide

**0.0f** by **0.0f**, as well as when calculating square root of a negative

number.

**Real Type Double**

**Real Floating-Point Types – Example**

Here is an example in which we declare variables of real number types, assign

values to them and print them:

float floatPI = 3.14f;

Console.WriteLine(floatPI); // 3.14

double doublePI = 3.14;

Console.WriteLine(doublePI); // 3.14

double nan = Double.NaN;

Console.WriteLine(nan); // NaN

double infinity = Double.PositiveInfinity;

Console.WriteLine(infinity); // Infinity

**Precision of the Real Types**

**Accuracy of Real Types – Example**

The real types in C# we went over – **float** and **double** – differ not only by

the range of possible values they can take, but also by their **precision** (the

number of decimal digits, which they can preserve). The first type has a

precision of 7 digits, the second – 15-16 digits.

// Declare some variables

float floatPI = 3.141592653589793238f;

double doublePI = 3.141592653589793238;

// Print the results on the console

Console.WriteLine("Float PI is: " + floatPI);

Console.WriteLine("Double PI is: " + doublePI);

// Console output:

// Float PI is: 3.141593

// Double PI is: 3.14159265358979

**About the Presentation of the Real Types**

Real floating-point numbers in C# consist of three components (according to

the standard IEEE 754): **sign** (1 or -1), **mantissa** and **order** (**exponent**),

and their values are calculated by a complex formula. More detailed

information about the representation of the real numbers is provided in the "Numeral Systems" where we will take an in-depth look at the

representation of numbers and other data types in computing.

**Errors in Calculations with Real Types**

In calculations with real floating-point data types it is possible to observe

**strange behavior**, because during the representation of a given real number

it often happens **to lose accuracy**. The reason for this is the inability of some

real numbers to be represented exactly as a sum of negative powers of the

number 2. Examples of numbers that do not have an accurate representation

in **float** and **double** types are for instance 0.1, 1/3, 2/7 and other. Here is a

sample C# code, which demonstrates errors in calculations with floating-point

numbers in C#:

float f = 0.1f;

Console.WriteLine(f); // 0.1 (correct due to rounding)

double d = 0.1f;

Console.WriteLine(d); // 0.100000001490116 (incorrect)

float ff = 1.0f / 3;

Console.WriteLine(ff); // 0.3333333 (correct due to rounding)

double dd = ff;

Console.WriteLine(dd); // 0.333333343267441 (incorrect)

**Boolean Type**

Boolean type is declared with the keyword **bool**. It has two possible values:

**true** and **false**. Its default value is **false**. It is used most often to store the

calculation **result of logical expressions**.

**Boolean Type – Example**

Consider an example in which we declare several variables from the already

known types, initialize them, compare them and print the result on the

console:

// Declare some variables

int a = 1;

int b = 2;

// Which one is greater?

bool greaterAB = (a > b);

// Is 'a' equal to 1?

bool equalA1 = (a == 1);

// Print the results on the console

if (greaterAB)

{

Console.WriteLine("A > B");

}

else

{

Console.WriteLine("A <= B");

}

Console.WriteLine("greaterAB = " + greaterAB);

Console.WriteLine("equalA1 = " + equalA1);

// Console output:

// A <= B

// greaterAB = False

// equalA1 = True

In the example above, we declare two variables of type **int**, compare them

and assign the result to the **bool** variable **greaterAB**. Similarly, we do the

same for the variable **equalA1**. If the variable **greaterAB** is **true**, then **A > B**

is printed on the console, otherwise **A <= B** is printed.

**Character Type**

Character type is a **single character** (16-bit number of a Unicode table

character). It is declared in C# with the keyword **char**. **The Unicode table** is

a technological standard that represents any character (letter, punctuation,

etc.) from all human languages as writing systems (all languages and

alphabets) with an integer or a sequence of integers. More about the **Unicode**

table can be found in the "Strings and Text Processing". The smallest

possible value of a **char** variable is **0**, and the largest one is 65535. The

values of type **char** are letters or other characters, and are enclosed in

apostrophes.

**Character Type – Example**

Consider an example in which we declare one variable of type **char**, initialize

it with value **'a'**, then **'b'**, then **'A'** and print the Unicode values of these

letters to the console:

// Declare a variable

char ch = 'a';

// Print the results on the console

Console.WriteLine(

"The code of '" + ch + "' is: " + (int)ch);

ch = 'b';

Console.WriteLine(

"The code of '" + ch + "' is: " + (int)ch);

ch = 'A';

Console.WriteLine(

"The code of '" + ch + "' is: " + (int)ch);

// Console output:

// The code of 'a' is: 97

// The code of 'b' is: 98

// The code of 'A' is: 65

**Strings**

**Strings are sequences of characters**. In C# they are declared by the

keyword **string**. Their default value is **null**. Strings are enclosed in quotation

marks. Various text-processing operations can be performed using strings:

concatenation (joining one string with another), splitting by a given separator,

searching, replacement of characters and others. More information about text

processing can be found in the "Strings and Text Processing", where

you will find detailed explanation on what a string is, what its applications are

and how we can use it.

**Strings – Example**

Consider an example in which we declare several variables of type **string**,

initialize them and print their values on the console:

// Declare some variables

string firstName = "John";

string lastName = "Smith";

string fullName = firstName + " " + lastName;

// Print the results on the console

Console.WriteLine("Hello, " + firstName + "!");

Console.WriteLine("Your full name is " + fullName + ".");

// Console output:

// Hello, John!

// Your full name is John Smith.

**Object Type**

Object type is a special type, which is the parent of all other types in the .NET

Framework. Declared with the keyword **object**, it can take values from **any**

**other type**. It is a reference type, i.e. an index (address) of a memory area

which stores the actual value.

**Using Objects – Example**

Consider an example in which we declare several variables of type **object**,

initialize them and print their values on the console:

// Declare some variables

object container1 = 5;

object container2 = "Five";

// Print the results on the console

Console.WriteLine("The value of container1 is: " + container1);

Console.WriteLine("The value of container2 is: " + container2);

// Console output:

// The value of container1 is: 5

// The value of container2 is: Five.

As you can see from the example, we can store the value of any other type in

an **object** type variable. This makes the **object** type a universal data

container.

**Nullable Types**

**Nullable types** are specific **wrappers** around the value types (as **int**,

**double** and **bool**) that allow storing data with a **null** value. This provides

opportunity for types that generally do not allow lack of value (i.e. value

**null**) to be used as reference types and to accept both normal values and the

special one **null**. Thus nullable types **hold an optional value**.

**Example**

int i = 5;

int? ni = i;

Console.WriteLine(ni); // 5

// i = ni; // this will fail to compile

Console.WriteLine(ni.HasValue); // True

i = ni.Value;

Console.WriteLine(i); // 5

ni = null;

Console.WriteLine(ni.HasValue); // False

//i = ni.Value; // System.InvalidOperationException

i = ni.GetValueOrDefault();

Console.WriteLine(i); // 0

The example above shows how a **nullable variable** (**int?**) can have a value

directly added even if the value is non-nullable (**int**). The opposite is not

directly possible. For this purpose, the nullable types’ property **Value** can be

used. It returns the value stored in the nullable type variable, or produces an

error (**InvalidOperationException**) during program execution if the value is

missing (**null**). In order to check whether a variable of nullable type has a

value assigned, we can use the Boolean property **HasValue**. Another useful

method is **GetValueOrDefault()**. If the nullable type variable has a value,

this method will return its value, else it will return the default value for the

nullable type (most commonly **0**).

Nullable types are used for storing information, which is **not mandatory**. For

example, if we want to store data for a student such as the first name and

last name as mandatory and age as not required, we can use type **int?** for

the age variable:

string firstName = "John";

string lastName = "Smith";

int? age = null;

**Operators**

**and Expressions**

**Arithmetical Operators – Example**

Here are some examples of arithmetic operators and their effect:

int squarePerimeter = 17;

double squareSide = squarePerimeter / 4.0;

double squareArea = squareSide \* squareSide;

Console.WriteLine(squareSide); // 4.25

Console.WriteLine(squareArea); // 18.0625

int a = 5;

int b = 4;

Console.WriteLine(a + b); // 9

Console.WriteLine(a + (b++)); // 9

Console.WriteLine(a + b); // 10

Console.WriteLine(a + (++b)); // 11

Console.WriteLine(a + b); // 11

Console.WriteLine(14 / a); // 2

Console.WriteLine(14 % a); // 4

int one = 1;

int zero = 0;

// Console.WriteLine(one / zero); // DivideByZeroException

double dMinusOne = -1.0;

double dZero = 0.0;

Console.WriteLine(dMinusOne / zero); // -Infinity

Console.WriteLine(one / dZero); // Infinity

The table and the following example show that the logical "AND" (**&&**) returns

true only when both variables contain truth. Logical "OR" (**||**) returns true

when at least one of the operands is true. The logical negation operator (**!**)

changes the value of the argument. For example, if the operand has a value

**true** and a negation operator is applied, the new value will be **false**. The

negation operator is a unary operator and it is placed before the argument.

Exclusive "OR" (**^**) returns **true** if only one of the two operands has the value

**true**. If the two operands have different values, exclusive "OR" will return the

result **true**, if they have the same values it will return **false**.

**Logical Operators – Example**

The following example illustrates the usage of the logical operators and their

actions:

bool a = true;

bool b = false;

Console.WriteLine(a && b); // False

Console.WriteLine(a || b); // True

Console.WriteLine(!b); // True

Console.WriteLine(b || true); // True

Console.WriteLine((5 > 7) ^ (a == b)); // False

**Laws of De Morgan**

Logical operations fall under the **laws of De Morgan** from the mathematical

logic:

!(a && b) == (!a || !b)

!(a || b) == (!a && !b)

The first law states that the negation of the conjunction (logical AND) of two

propositions is equal to the disjunction (logical OR) of their negations.

The second law states that the negation of the disjunction of both statements

is equivalent to the conjunction of their negations.

**Bitwise Operators – Example**

Here is an example of using bitwise operators. The binary representation of

the numbers and the results of the bitwise operators are shown in the

byte a = 3; // 0000 0011 = 3

byte b = 5; // 0000 0101 = 5

Console.WriteLine(a | b); // 0000 0111 = 7

Console.WriteLine(a & b); // 0000 0001 = 1

Console.WriteLine(a ^ b); // 0000 0110 = 6

Console.WriteLine(~a & b); // 0000 0100 = 4

Console.WriteLine(a << 1); // 0000 0110 = 6

Console.WriteLine(a << 2); // 0000 1100 = 12

Console.WriteLine(a >> 1); // 0000 0001 = 1

**Comparison Operators**

Comparison operators in C# are used to compare two or more operands. C#

supports the following comparison operators:

- greater than (**>**)

- less than (**<**)

- greater than or equal to (**>=**)

- less than or equal to (**<=**)

- equality (**==**)

- difference (**!=**)**148**

All comparison operators in C# are **binary** (take two operands) and the

returned result is a Boolean value (**true** or **false**). Comparison operators

have lower priority than arithmetical operators but higher than the

assignment operators.

**Comparison Operators – Example**

The following example demonstrates the usage of comparison operators in

C#:

int x = 10, y = 5;

Console.WriteLine("x > y : " + (x > y)); // True

Console.WriteLine("x < y : " + (x < y)); // False

Console.WriteLine("x >= y : " + (x >= y)); // True

Console.WriteLine("x <= y : " + (x <= y)); // False

Console.WriteLine("x == y : " + (x == y)); // False

Console.WriteLine("x != y : " + (x != y)); // True

**Assignment Operators – Example**

Here is an example to show the usage of the assignment operator:

int x = 6;

string helloString = "Hello string.";

int y = x;

**Cascade Assignment**

The assignment operator can be used in **cascade** (more than once in the

same expression). In this case assignments are carried out consecutively from

right to left. Here’s an example:

int x, y, z;

x = y = z = 25;

**Compound Assignment Operators**

Except the assignment operator there are also **compound assignment**

**operators**. They help to reduce the volume of the code by typing two

operations together with an operator: operation and assignment. Compound

operators have the following syntax:

operand1 operator = operand2;

The upper expression is like the following:

operand1 = operand1 operator operand2;

Here is an example of a compound operator for assignment:

int x = 2;

int y = 4;

x \*= y; // Same as x = x \* y;

Console.WriteLine(x); // 8

int x = 6;

int y = 4;

Console.WriteLine(y \*= 2); // 8

int z = y = 3; // y=3 and z=3

Console.WriteLine(z); // 3

Console.WriteLine(x |= 1); // 7

Console.WriteLine(x += 3); // 10

Console.WriteLine(x /= 2); // 5

**Conditional Operator "?:" – Example**

The following example shows the usage of the operator "**?:**":

int a = 6;

int b = 4;

Console.WriteLine(a > b ? "a>b" : "b<=a"); // a>b

int num = a == b ? 1 : -1; // num will have value -1

**Implicit Type Conversion**

Implicit (hidden) type conversion is possible only when there is no risk of data

loss during the conversion, i.e. when converting from a lower range type to a

larger range (e.g. from **int** to **long**). To make an implicit conversion it is not

necessary to use any operator and therefore such transformation is called

implicit. The **implicit conversion** is done automatically by the compiler when

you assign a value with lower range to a variable with larger range or if the

expression has several types with different ranges. In such case the

conversion is executed into the type with the highest range.

**Implicit Type Conversion**

Implicit (hidden) type conversion is possible only when there is no risk of data

loss during the conversion, i.e. when converting from a lower range type to a

larger range (e.g. from **int** to **long**). To make an implicit conversion it is not

necessary to use any operator and therefore such transformation is called

implicit. The **implicit conversion** is done automatically by the compiler when

you assign a value with lower range to a variable with larger range or if the

expression has several types with different ranges. In such case the

conversion is executed into the type with the highest range.

**– Examples**

Here is an example of implicit type conversion:

int myInt = 5;

Console.WriteLine(myInt); // 5

long myLong = myInt;

Console.WriteLine(myLong); // 5

Console.WriteLine(myLong + myInt); // 10

**Explicit Type Conversion**

Explicit type conversion is used whenever there is a possibility of data loss.

When converting floating point type to integer type there is always a loss of

data coming from the elimination of the fractional part and an **explicit**

**conversion** is obligatory (e.g. **double** to **long**). To make such a conversion it

is necessary to use the operator for data conversion (**type**). There may also

be data loss when converting a type with a wider range to type with a

narrower one (**double** to **float** or **long** to **int**)

**– Example**

The following example illustrates the use of explicit type conversion and data

loss that may occur in some cases:

double myDouble = 5.1d;

Console.WriteLine(myDouble); // 5.1

long myLong = (long)myDouble;

Console.WriteLine(myLong); // 5

myDouble = 5e9d; // 5 \* 10^9

Console.WriteLine(myDouble); // 5000000000

int myInt = (int)myDouble;

Console.WriteLine(myInt); // -2147483648

Console.WriteLine(int.MinValue); // -2147483648

**Conditional**

**Statements**

**Comparison operators** can be used to compare expressions such as two

numbers, two numerical expressions, or a number and a variable. The result

of the comparison is a Boolean value (**true** or **false**).

Let’s look at an example of using comparisons:

int weight = 700;

Console.WriteLine(weight >= 500); // True

char gender = 'm';

Console.WriteLine(gender <= 'f'); // False

double colorWaveLength = 1.630;

Console.WriteLine(colorWaveLength > 1.621); // True

int a = 5;

int b = 7;

bool condition = (b > a) && (a + b < a \* b);

Console.WriteLine(condition); // True

Console.WriteLine('B' == 'A' + 1); // True

**Comparison of Integers and Characters**

When comparing integers and characters, we directly compare their binary

representation in memory i.e. we **compare their values**. For example, if we

compare two numbers of type **int**, we will compare the values of their

respective series of 4 bytes. Here is one example for integer and character

comparisons:

Console.WriteLine("char 'a' == 'a'? " + ('a' == 'a')); // True

Console.WriteLine("char 'a' == 'b'? " + ('a' == 'b')); // False

Console.WriteLine("5 != 6? " + (5 != 6)); // True

Console.WriteLine("5.0 == 5L? " + (5.0 == 5L)); // True

Console.WriteLine("true == false? " + (true == false)); // False

**Logical Operators**

Let’s recall the logical operators in C#. They are often used to construct

logical (Boolean) expressions. The logical operators are: **&&**, **||**, **!** and **^**.

**Logical Operators && and ||**

The logical operators **&&** (logical AND) and **||** (logical OR) are only used on

Boolean expressions (values of type **bool**).

bool result = (2 < 3) && (3 < 4);

Similarly, the operator || returns true if at least one of the two operands has

the value "true". Example:

bool result = (2 < 3) || (1 == 2);

**Logical Operators & and |**

The operators for comparison **&** and **|** are similar to **&&** and **||**, respectively.

The difference lies in the fact that both operands are calculated one after the

other, although the final result is known in advance.

**Logical Operators ^ and !**

The **^** operator, also known as **exclusive OR (XOR)**, belongs to the full

circuit operators, because both operands are calculated one after the other.

The result of applying the operator is **true if exactly one of the operands**

**is true, but not both simultaneously**. Otherwise the result is **false**. Here

is an example:

Console.WriteLine("Exclusive OR: "+ ((2 < 3) ^ (4 > 3)));

The operator **!** returns the reversed value of the Boolean expression to

which it is attached. Example:

bool value = !(7 == 5); // True

Console.WriteLine(value);

**Conditional Statements "if" and "if-else"**

After reviewing how to compare expressions, we will continue with conditional

statements, which will allow us to implement programming logic.

**Conditional statements if** and **if-else** are conditional control statements.

Because of them the program can behave differently based on a defined

condition checked during the execution of the statement.

**Conditional Statement "if"**

The main format of the conditional statements **if** is as follows:

if (Boolean expression)

{

Body of the conditional statement;

}

**Conditional Statement "if" – Example**

Let’s take a look at an example of using a conditional statement **if**:

static void Main()

{

Console.WriteLine("Enter two numbers.");

Console.Write("Enter first number: ");

int firstNumber = int.Parse(Console.ReadLine());

Console.Write("Enter second number: ");

int secondNumber = int.Parse(Console.ReadLine());

int biggerNumber = firstNumber;

if (secondNumber > firstNumber)

{

biggerNumber = secondNumber;

}

Console.WriteLine("The bigger number is: {0}", biggerNumber);

}

**Conditional Statement "if-else"**

In C#, as in most of the programming languages there is a conditional

statement with **else** clause: the **if**-**else** statement. Its format is the

following:

if (Boolean expression)

{

Body of the conditional statement;

}

else

{

Body of the else statement;

}

**Conditional Statement "if-else" – Example**

Let’s take a look at the next example and illustrate how the **if-else**

statement works:

static void Main()

{

int x = 2;

if (x > 3)

{

Console.WriteLine("x is greater than 3");

}

else

{

Console.WriteLine("x is not greater than 3");

}

}

**Nested "if" Statements – Example**

Here is an example of using nested **if** structures:

int first = 5;

int second = 3;

if (first == second)

{

Console.WriteLine("These two numbers are equal.");

}

else

{

if (first > second)

{

Console.WriteLine("The first number is greater.");

}

else

{

Console.WriteLine("The second number is greater.");

}

}

**Sequences of "if-else-if-else-…"**

Sometimes we need to use a **sequence of if structures**, where the **else**

clause is a new **if** structure. If we use nested **if** structures, the code would

be pushed too far to the right. That’s why in such situations it is allowed to

use a new **if** right after the **else**. It’s even considered a good practice. Here

is an example:

char ch = 'X';

if (ch == 'A' || ch == 'a')

{

Console.WriteLine("Vowel [ei]");

}

else if (ch == 'E' || ch == 'e')

{

Console.WriteLine("Vowel [i:]");

}

else if (ch == 'I' || ch == 'i')

{

Console.WriteLine("Vowel [ai]");

}

else if (ch == 'O' || ch == 'o')

{

Console.WriteLine("Vowel [ou]");

}

else if (ch == 'U' || ch == 'u')

{

Console.WriteLine("Vowel [ju:]");

}

else

{

Console.WriteLine("Consonant");

}

**Conditional Statement "switch-case"**

In the following section we will cover the conditional statement **switch**. It is

used for choosing among a list of possibilities.

**How Does the "switch-case" Statement Work?**

The structure **switch-case** chooses which part of the programming code to

execute based on the calculated value of a certain expression (most often of

integer type). The format of the structure for choosing an option is as follows:

switch (integer\_selector)

{

case integer\_value\_1:

statements;

break;

case integer\_value\_2:

statements;

break;

// …

default:

statements;

break;

}

**Loops**

**While Loops**

One of the simplest and most commonly used loops is **while**.

while (condition)

{

loop body;

}

Example-

// Initialize the counter

int counter = 0;

// Execute the loop body while the loop condition holds

while (counter <= 9)

{

// Print the counter value

Console.WriteLine("Number : " + counter);

// Increment the counter

counter++;

}

**Summing the Numbers from 1 to N**

In this example we will examine how by using the **while loop** we can find the

sum of the numbers from **1** to **n**. The number **n** is read from the console:

Console.Write("n = ");

int n = int.Parse(Console.ReadLine());

int num = 1;

int sum = 1;

Console.Write("The sum 1");

while (num < n)

{

num++;

sum += num;

Console.Write(" + " + num);

}

Console.WriteLine(" = " + sum);

**Check If a Number Is Prime – Example**

We will write a **program to check whether a given number is prime or**

**not**. We will read the number to check from the console. As we know from the

mathematics, a prime number is any positive integer number, which, is not

divisible by any other numbers except 1 and itself. We can check if the

number **num** is prime when in a loop we check if it divides by numbers from 2

to **√num**:

Console.Write("Enter a positive number: ");

int num = int.Parse(Console.ReadLine());

int divider = 2;

int maxDivider = (int)Math.Sqrt(num);

bool prime = true;

while (prime && (divider <= maxDivider))

{

if (num % divider == 0)

{

prime = false;

}

divider++;

}

Console.WriteLine("Prime? " + prime);

**Do-While Loops**

The **do-while** loop is used when we want to guarantee that the sequence of

operations in it will be executed repeatedly and at least once in the beginning

of the loop.

**Calculating Factorial – Example**

In this example we will again calculate the factorial of a given number n, but

this time instead of an infinite **while** loop we will use a **do-while**. The logic is

similar to that in the previous example:

Console.Write("n = ");

int n = int.Parse(Console.ReadLine());

decimal factorial = 1;

do

{

factorial \*= n;

n--;

} while (n > 0);

Console.WriteLine("n! = " + factorial);

**For-Loop – Example**

Here is a complete example of a **for**-loop:

for (int i = 0; i <= 10; i++)

{

Console.Write(i + " ");

}

**Foreach Loops**

Here is how a **foreach** loop looks like:

foreach (type variable in collection)

{

statements;

}

As we see, **it is significantly simpler than the standard for-loop** and

therefore is very often preferred by developers because it saves writing when

you need to go through all the elements of a given collection. Here is an

example that shows how we can use **foreach**:

int[] numbers = { 2, 3, 5, 7, 11, 13, 17, 19 };

foreach (int i in numbers)

{

Console.Write(" " + i);

}

Console.WriteLine();

string[] towns = { "London", "Paris", "Milan", "New York" };

foreach (string town in towns)

{

Console.Write(" " + town);

}

**Nested Loops**

The **nested loops** are programming constructs consisting of several loops

located into each other. The innermost loop is executed more times, and the

outermost – less times. Let’s see how two nested loops look like:

for (initialization, verification, update)

{

for (initialization, verification, update)

executable code

}

}

**Printing a Triangle – Example**

int n = int.Parse(Console.ReadLine());

for (int row = 1; row <= n; row++)

{

for (int col = 1; col <= row; col++)

{

Console.Write(col + " ");

}

Console.WriteLine();

}

**Prime Numbers in an Interval – Example**

Let’s consider another **example of nested loops**. We set a goal to print on

the console all prime number in the interval [n…m]. We will limit the interval

by a **for**-loop and in order to check for a prime number we will use a nested

**while** loop:

Console.Write("n = ");

int n = int.Parse(Console.ReadLine());

Console.Write("m = ");

int m = int.Parse(Console.ReadLine());

for (int num = n; num <= m; num++)

{

bool prime = true;

int divider = 2;

int maxDivider = (int)Math.Sqrt(num);

while (divider <= maxDivider)

{

if (num % divider == 0)

{

prime = false;

break;

}

divider++;

}

if (prime)

{

Console.Write(" " + num);

}

}

**Lucky Numbers – Example**

Let’s consider another example through which we will show that we can put

even **more than two loops into each other**. Our purpose is to find and

print all four-digit numbers of the type ABCD, where: A+B = C+D (we call

them lucky numbers). We will implement it with four **for**-loops – one for each

digit. The outermost loop will define the thousands. It will start from 1 and

the rest of the loops – from 0. They will determine the hundreds, the tens and

the units. We will perform a check if our current number in the most inner

loop is a lucky one and if so, we will print it on the console. Here is an

implementation example:

for (int a = 1; a <= 9; a++)

{

for (int b = 0; b <= 9; b++)

{

for (int c = 0; c <= 9; c++)

{

for (int d = 0; d <= 9; d++)

{

if ((a + b) == (c + d))

{

Console.WriteLine(

" " + a + " " + b + " " + c + " " + d);

}

}

}

}

}

**Arrays**

**Declaring an Array**

We declare an array in C# in the following way:

int[] myArray;

**Declaration and Initialization of an Array – Example**

Here is one more example how to **declare and initialize an array**:

string[] daysOfWeek =

{ "Monday", "Tuesday", "Wednesday","Thursday", "Friday",

"Saturday", "Sunday" };

Here is an example, where we allocate an array of numbers and then we

change some of them:

int[] myArray = new int[6];

myArray[1] = 1;

myArray[5] = 5;

int[] arr = new int[5];

for (int i = 0; i < arr.Length; i++)

{

arr[i] = i;

}

**IndexOutOfRangeExample.cs**

class IndexOutOfRangeExample

{

static void Main()

{

int[] myArray = { 1, 2, 3, 4, 5, 6 };

Console.WriteLine(myArray[6]);

}

}

**Reversing an Array – Example**

In the next example we will access elements and change them using their

indices. The task is to print the elements in **reversed order**. We will reverse

the elements of the array using a second, auxiliary array, where we will keep

the elements of the first one, but in a reversed order. Note that the length of

both arrays is the same and it stays unchanged after the first allocation:

**ArrayReverseExample.cs**

class ArrayReverseExample

{

static void Main()

{

int[] array = { 1, 2, 3, 4, 5 };

// Get array size

int length = array.Length;

// Declare and create the reversed array

int[] reversed = new int[length];

// Initialize the reversed array

for (int index = 0; index < length; index++)

{

reversed[length - index - 1] = array[index];

}

// Print the reversed array

for (int index = 0; index < length; index++)

{

Console.Write(reversed[index] + " ");

}

}

}

// Output: 5 4 3 2 1

**Reading an Array from the Console**

Let’s see how we can read values of an array from the console. We will use a

**for**-loop and the .NET Framework tools for reading from the console.

Initially we read a line from the console using **Console.ReadLine()**, and then

we parse that line to an integer number using **int.Parse()** and we set it to

the variable **n**. We then use the number **n** as length of the array.

int n = int.Parse(Console.ReadLine());

int[] array = new int[n];

for (int i = 0; i < n; i++)

{

array[i] = int.Parse(Console.ReadLine());

}

Console.Write("Enter a positive integer: ");

int n = int.Parse(Console.ReadLine());

int[] array = new int[n];

Console.WriteLine("Enter the values of the array:");

for (int i = 0; i < n; i++)

{

array[i] = int.Parse(Console.ReadLine());

}

bool symmetric = true;

for (int i = 0; i < array.Length / 2; i++)

{

if (array[i] != array[n - i - 1])

{

symmetric = false;

break;

}

}

Console.WriteLine("Is symmetric? {0}", symmetric);

**Iteration with a For Loop**

It is a good practice to use **for**-loops, when we work with arrays and

structures with indices. In the following example we will double the values of

all elements of an array of numbers and we will print them:

int[] array = new int[] { 1, 2, 3, 4, 5 };

Console.Write("Output: ");

for (int index = 0; index < array.Length; index++)

{

// Doubling the number

array[index] = 2 \* array[index];

// Print the number

Console.Write(array[index] + " ");

}

// Output: 2 4 6 8 10

**Recursion**

**What Is Recursion?**

We call an object **recursive** if it contains itself, or if it is defined by itself.

**Recursion** is a programming technique in which **a method makes a call to**

**itself** to solve a particular problem. Such methods are called **recursive**.

static long Fib(int n)

{

if (n <= 2)

{

return 1;

}

return Fib(n - 1) + Fib(n - 2);

}

**RecursiveFactorial.cs**

using System;

class RecursiveFactorial

{

static void Main()

{

Console.Write("n = ");

int n = int.Parse(Console.ReadLine());

decimal factorial = Factorial(n);

Console.WriteLine("{0}! = {1}", n, factorial);

}

static decimal Factorial(int n)

{

// The bottom of the recursion

if (n == 0)

{

return 1;

}

// Recursive call: the method calls itself

else

{

return n \* Factorial(n - 1);

}

}

}

**recursive nested loops** solution:

**RecursiveNestedLoops.cs**

using System;

class RecursiveNestedLoops

{

static int numberOfLoops;

static int numberOfIterations;

static int[] loops;

static void Main()

{

Console.Write("N = ");

numberOfLoops = int.Parse(Console.ReadLine());

Console.Write("K = ");

numberOfIterations = int.Parse(Console.ReadLine());

loops = new int[numberOfLoops];

NestedLoops(0);

}

static void NestedLoops(int currentLoop)

{

if (currentLoop == numberOfLoops)

{

PrintLoops();

return;

}

for (int counter=1; counter<=numberOfIterations; counter++)

{

loops[currentLoop] = counter;

NestedLoops(currentLoop + 1);

}

}

static void PrintLoops()

{

for (int i = 0; i < numberOfLoops; i++)

{

Console.Write("{0} ", loops[i]);

}

Console.WriteLine();

}

}

**ps algorithm**:

**IterativeNestedLoops.cs**

using System;

class IterativeNestedLoops

{

static int numberOfLoops;

static int numberOfIterations;

static int[] loops;

static void Main()

{

Console.Write("N = ");

numberOfLoops = int.Parse(Console.ReadLine());

Console.Write("K = ");

numberOfIterations = int.Parse(Console.ReadLine());

loops = new int[numberOfLoops];

NestedLoops();

}

static void NestedLoops()

{

InitLoops();

int currentPosition;

while (true)

{

PrintLoops();

currentPosition = numberOfLoops - 1;

loops[currentPosition] = loops[currentPosition] + 1;

while (loops[currentPosition] > numberOfIterations)

{

loops[currentPosition] = 1;

currentPosition--;

if (currentPosition < 0)

{

return;

}

loops[currentPosition] = loops[currentPosition] + 1;

}

}

}

static void InitLoops()

{

for (int i = 0; i < numberOfLoops; i++)

{

loops[i] = 1;

}

}

static void PrintLoops()

{

for (int i = 0; i < numberOfLoops; i++)

{

Console.Write("{0} ", loops[i]);

}

Console.WriteLine();

}

}

**Object-oriented programming (OOP)**

**Classes and Objects**

Over the last few decades programming and informatics have experienced

incredible growth and concepts, which have changed the way programs, are

built. **Object-oriented programming (OOP)** introduces such radical idea.

We are going to make a short introduction to the principles of OOP and the

concepts used in it. Firstly, we are going to explain what classes and objects

are. These two terms are basic for OOP and inseparable part from the life of

any modern programmer.

**What Is Object-Oriented Programming?**

Object-oriented programming (OOP) is a programming paradigm, which uses

**objects** and their interactions for building computer programs. Thus an easy

to understand, simple model of the subject area is achieved, which gives an

opportunity to the programmer to solve intuitively (by simple logic) many of

the problems, which occur in the real world.

For now we are not going to get into details what the goals and the

advantages of OOP are, as well as explaining in details the principles for

building hierarchies of classes and objects. We are going to mention only that

programming techniques of OOP often include **encapsulation**, **abstraction**,

**polymorphism** and **inheritance**. we are going to consider them later in the

"Principles of Object-Oriented Programming".

**An Example Class**

We are going to give an example of a class in C#, which contains the listed

elements. The class **Cat** models the real-world object "cat" and has the

properties **name** and **color**. The given class defines several fields, properties

and methods, which we are going to use later. You can now see the definition

of the class (we are not going to consider in details the definition of the

classes – we are going to focus on that in the "Defining Classes"):

public class Cat

{

// Field name

private string name;

// Field color

private string color;

public string Name

{

// Getter of the property "Name"

get

{

return this.name;

}

// Setter of the property "Name"

set

{

this.name = value;

}

}

public string Color

{

// Getter of the property "Color"

get

{

return this.color;

}

// Setter of the property "Color"

set

{

this.color = value;

}

}

// Default constructor

public Cat()

{

this.name = "Unnamed";

this.color = "gray";

}

// Constructor with parameters

public Cat(string name, string color)

{

this.name = name;

this.color = color;

}

// Method SayMiau

public void SayMiau()

{

Console.WriteLine("Cat {0} said: Miauuuuuu!", name);

}

}

static void Main()

{

Cat firstCat = new Cat();

firstCat.Name = "Tony";

firstCat.SayMiau();

Cat secondCat = new Cat("Pepy", "red");

secondCat.SayMiau();

Console.WriteLine("Cat {0} is {1}.",

secondCat.Name, secondCat.Color);

}

**Calling Methods of Objects – Example**

We are going to complement the example we already gave as we call the

method **SayMiau** of the class **Cat**. Here is the result:

class CatManipulating

{

static void Main()

{

Cat myCat = new Cat();

myCat.Name = "Alfred";

Console.WriteLine("The name of my cat is {0}.",myCat.Name);

**myCat.SayMiau();**

}

}

**Calling Constructors – Example**

Lets' take a look again at the definition of the class **Cat** and more particularly

at the two constructors of the class:

public class Cat

{

// Field name

private string name;

// Field color

private string color;

…

// Parameterless constructor

public Cat()

{

this.name = "Unnamed";

this.color = "gray";

}

// Constructor with parameters

public Cat(string name, string color)

{

this.name = name;

this.color = color;

}

}

{

static void Main()

{

Cat someCat = new Cat();

someCat.SayMiau();

Console.WriteLine("The color of cat {0} is {1}.", someCat.Name, someCat.Color);

Cat someCat = new Cat("Johnny", "brown");

someCat.SayMiau();

Console.WriteLine("The color of cat {0} is {1}.",

someCat.Name, someCat.Color);

}

}

**Static Fields and Methods – Example**

public class Sequence

{

// Static field, holding the current sequence value

private static int currentValue = 0;

// Intentionally deny instantiation of this class

private Sequence()

{

}

// Static method for taking the next sequence value

public static int NextValue()

{

currentValue++;

return currentValue;

}

}

class SequenceManipulating

{

static void Main()

{

Console.WriteLine("Sequence[1...3]: {0}, {1}, {2}",

Sequence.NextValue(), Sequence.NextValue(),

Sequence.NextValue());

}

}

**What Are Namespaces in C#?**

**Namespaces** in C# are **named groups of classes**, which are logically

related without any specific requirement on how to be placed in the file

system. However, it is considered that the folder name should match the

namespace name and the names of the files should match the names of the

classes, which are defined in them. We have to note that in some

programming languages the compilation of the source code in a given

namespace depends on the distribution of the elements of the namespace in

folders and files on the disk. In Java, for instance, the described file

organization is mandatory (if it is not followed, compilation errors occur). C#

is not so strict regarding this.

Now, let’s consider the mechanism for defining namespaces.

**Defining Namespaces**

In case we like to create a new namespace or a new class which belongs to a

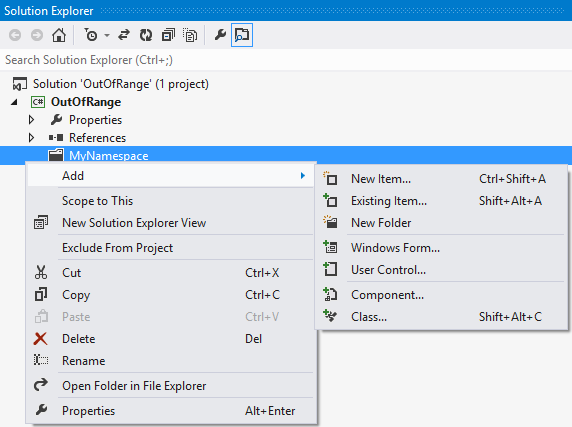
given namespace, in Visual Studio this happens automatically by the

commands in the context menu of the Solution Explorer (on right click on the

corresponding folder). By default the Solution Explorer is visualized like a

Dock in the right part of the integrated environment. We are going to

illustrate how we could add a new class in the already existing namespace



As the project is called **MyConsoleApplication** and we are adding in its folder

**MyNamespace**, the newly created class is going to be in the following

namespace:

namespace MyConsoleApplication.MyNamespace

**Using a Namespace – Example**

In order to illustrate the principle of inclusion of a namespace, we are going to

consider the following program which reads numbers, saves them in lists and

counts how many of them are integer numbers and how many are double:

class NamespaceImportTest

{

static void Main()

{

System.Collections.Generic.List<int> ints =

new System.Collections.Generic.List<int>();

System.Collections.Generic.List<double> doubles =

new System.Collections.Generic.List<double>();

while (true)

{

int intResult;

double doubleResult;

Console.WriteLine("Enter an int or a double:");

string input = Console.ReadLine();

if (int.TryParse(input, out intResult))

{

ints.Add(intResult);

}

else if (double.TryParse(input, out doubleResult))

{

doubles.Add(doubleResult);

}

else

{

break;

}

}

Console.Write("You entered {0} ints:", ints.Count);

foreach (var i in ints)

{

Console.Write(" " + i);

}

Console.WriteLine();

Console.Write("You entered {0} doubles:", doubles.Count);

foreach (var d in doubles)

{

Console.Write(" " + d);

}

Console.WriteLine();

}

}

**Exception**

**Handling**

**An Example Code Throwing an Exception**

Here is an example for a code that will **throw an exception**:

class ExceptionsDemo

{

static void Main()

{

string fileName = "WrongTextFile.txt";

ReadFile(fileName);

}

static void ReadFile(string fileName)

{

TextReader reader = **new StreamReader(fileName);**

string line = **reader.ReadLine();**

Console.WriteLine(line);

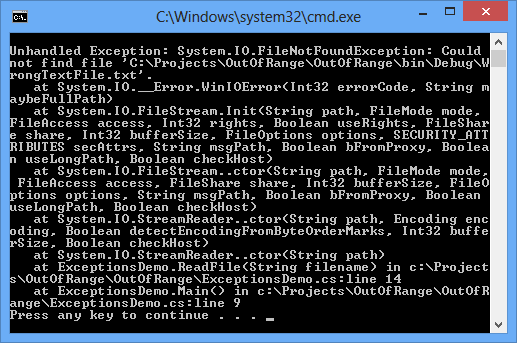
reader.Close();

}

}

his program will compile successfully but if you run it, the result will look like

the following (**FileNotFoundException** dumped on the console):



**The try-catch Programming Construct**

To handle an exception, we must surround the code that could throw an

exception with a **try-catch** block:

try

{

// Some code that may throw an exception

}

catch (ExceptionType objectName)

{

// Code handling an Exception

}

catch (ExceptionType objectName)

{

// Code handling an Exception

}

**Catching Exceptions – Example**

Let’s now modify the code in our previous example to make it handle its

exceptions. To do this, we wrap the code that could create problems in **try**

**catch** and then we add catch blocks to handle the two types of exceptions we

know could arise.

static void ReadFile(string fileName)

{

// Exceptions could be thrown in the code below

try

{

TextReader reader = new StreamReader(fileName);

string line = reader.ReadLine();

Console.WriteLine(line);

reader.Close();

}

catch (FileNotFoundException fnfe)

{

// Exception handler for FileNotFoundException

// We just inform the user that there is no such file

Console.WriteLine(

"The file '{0}' is not found.", fileName);

}

catch (IOException ioe)

{

// Exception handler for other input/output exceptions

// We just print the stack trace on the console

Console.WriteLine(ioe.StackTrace);

}

}

Now our method works in a different way. When **FileNotFoundException** is

thrown during the **StreamReader** initialization when executing the constructor

**new StreamReader(filename)**, the CLR will not execute the following lines

but will jump to the row where we catch the exception **catch**

**(FileNotFoundException fnfe)**:

catch (FileNotFoundException fnfe)

{

// Exception handler for FileNotFoundException

// We just inform the user that there is no such file

Console.WriteLine("The file '{0}' is not found.", fileName);

}

In our example, users will simply be informed that such file does not exist by

a message printed on the standard output:

The file 'WrongTextFile.txt' is not found.

Similarly, if an **IOException** is thrown during **reader.ReadLine()**, it is

handled by the block below:

catch (IOException ioe)

{

// Exception handler for FileNotFoundException

// We just print the stack trace on the screen

Console.WriteLine(ioe.StackTrace);

}

C# Lab content(in cdac)

**Day**1

**Eg.1:**

namespace Nm1

{

internal class CMain1

{

static void Main(string[] args)

{

Console.Write("Enter first number : ");

string? numStr=Console.ReadLine();

int x=int.Parse(numStr??"100");

Console.Write("Enter second number : ");

int y=int.Parse(Console.ReadLine()??"0");

while (x != y)

{

if(x>y)

x=x-y;

if(y>x)

y=y-x;

}

Console.WriteLine("HCF : is {0}", x);

}

}

}

**Eg. 2:**

namespace Nm2

{

internal class CMain2

{

static void Main1(string[] args)

{

for (int i = 1,sum=0,n=6; i < n; i++)

{

Console.WriteLine($"{sum = sum \* 10 + i}");

}

}

static void Main2(string[] args)

{

for (int i = 1, sum = 0, n = 6; i < n; i++)

{

Console.WriteLine($"{(sum = sum \* 10 + 1)\*i}");

}

}

static void Main(string[] args)

{

int x = 10;

int y = 20;

Console.WriteLine($"The sum of {x} and {y} is {x+y}");

}

}

}

**Eg.3:**

namespace Nm3

{

internal class CMain

{

static void Main(string[] args)

{

Console.WriteLine("For Each Loops");

string[] myPartners = {"Yuvi","Siddharth","Ranbir","Ranveer" };

int[] numYrs = { 2, 1, 3, 8 };

foreach (string partner in myPartners)

{

Console.WriteLine($"\t=>>{partner}");

}

foreach (var item in numYrs)

{

Console.WriteLine($"\t=>>{item}");

}

}

}

}

**Eg.4:**

namespace Nm4

{

internal class CMain

{

static void Main1(string[] args)

{

string st = new string('-', 50);

Console.WriteLine(st);

Console.WriteLine($"{new string('=',20)} If Statements {new string('=', 20)}");

//object obj1 = "123";

object obj1 = 123;

if(obj1 is int myVal1)

{

Console.WriteLine($"Apple {myVal1}");

}

if (obj1 is string myVal2)

{

Console.WriteLine($"Orange {myVal2}");

}

}

static void Main(string[] args)

{

object obj1 = 123;

int k = 10;

//Type ty = k.GetType();

Type ty = typeof(int); //"sachin".GetType();

if(ty is Type t )

{

Console.WriteLine($"India Won the WC [{t}]");

}

char c = '.';

if( c is >='a' and <='z' )

{

Console.WriteLine($"{c} is small alphabet");

}

if (c is (>= 'a' and <= 'z') or '.' or ',')

{

Console.WriteLine($"{c} is small alphabet or seperator");

}

}

}

}

**Eg.5:**

namespace Nm5

{

internal class CMain

{

static void Main(string[] args)

{

string line = new string('=', 20);

Console.WriteLine($"{line} Conditional Operator {line}");

string data = "We are World Champions";

Console.WriteLine(data.Length>0?"Apple":"Orange");

Console.WriteLine($"{line} Conditional ref Operator {line}");

string [] arr1= { "Yuvi", "Siddharth", "Ranbir", "Ranveer" };

string[] arr2 = { "Shahid", "Saif" };

foreach (var item in arr1)

{

Console.Write($"\t{item}");

}

Console.WriteLine();

foreach (var item in arr2)

{

Console.Write($"\t{item}");

}

Console.WriteLine();

int index = 1;

// ref string val =ref arr1[index];

ref string val = ref (index < 2) ? ref arr2[index] : ref arr1[index];

val = "Khan";

// Console.WriteLine(val);

foreach (var item in arr1)

{

Console.Write($"\t{item}");

}

Console.WriteLine();

foreach (var item in arr2)

{

Console.Write($"\t{item}");

}

Console.WriteLine();

}

}

}

**Day2**

**Eg.1:**

using DacLibrary;

namespace DacConsole

{

internal class Program

{

static void Main(string[] args)

{

Employee emp1 = CFactory.GetEmp("101", "sachin", "dev");

Console.WriteLine($"emp1 id = {emp1.ID}");

emp1.Name = "Tendulkar";

string h = emp1.Name;

emp1.DoEmpJob();

Employee emp2 = CFactory.GetEmp("102", "saurav", "tes");

emp2.DoEmpJob();

Employee emp3 = CFactory.GetEmp("103", "rahul", "arc");

emp3.DoEmpJob();

}

}

}

**Eg.2:**

namespace Question1

{

abstract class MyArray

{

int[] arr ={ 30,10,20,60,50,40};

public void Display()

{

Array.ForEach(arr,x => Console.Write($"{x,-15}"));

//foreach (var item in arr)

//{

// Console.Write($"{item,-15}");

//}

Console.WriteLine();

}

protected abstract bool Compare(int x, int y);

public void Sort()

{

for (int write = 0; write < arr.Length; write++)

{

for (int sort = 0; sort < arr.Length - 1; sort++)

{

if (Compare(arr[sort] , arr[sort + 1]))

{

int temp = arr[sort + 1];

arr[sort + 1] = arr[sort];

arr[sort] = temp;

}

}

}

}

}

class MyArrayAsc : MyArray

{

protected override bool Compare(int x, int y)

{

return x > y;

}

}

class MyArrayDesc : MyArray

{

protected override bool Compare(int x, int y)

{

return x < y;

}

}

public class CMain

{

static void Main(string[] args)

{

MyArrayAsc arr = new MyArrayAsc();

arr.Display();

arr.Sort();

arr.Display();

Console.WriteLine(new string('\_', 80));

MyArrayDesc arr1 = new MyArrayDesc();

arr1.Display();

arr1.Sort();

arr1.Display();

}

}

}

**Eg. 3:**

using System.Diagnostics.Metrics;

namespace DacLibrary

{

public abstract class Employee

{

string e\_id = string.Empty;

string e\_name = string.Empty;

private string e\_type = string.Empty;

private double e\_salary = 0.0;

public Employee(string id,string name)

{

e\_id = id;

e\_name = name;

}

//property

public string ID

{

get{ return e\_id; }

set{ e\_id = value; }

}

public string Name{get => e\_name;set => e\_name = value;}

protected string EType { get => e\_type; set => e\_type = value; }

protected double ESalary { get => e\_salary; set => e\_salary = value; }

protected abstract void ProcessSalary();

public void DoEmpJob()

{

Console.WriteLine("Employee Job Started");

ProcessSalary();

Console.WriteLine("Employee Job Completed");

Console.WriteLine(new String('\_',50));

}

}

internal class Developer : Employee

{

public Developer(string id, string name) : base(id, name)

{

}

protected override void ProcessSalary()

{

Console.WriteLine("Developer Salary Processed");

}

}

internal class Tester : Employee

{

public Tester(string id, string name) : base(id, name)

{

}

protected override void ProcessSalary()

{

Console.WriteLine("Tester Salary Processed");

}

}

internal class Architect : Employee

{

public Architect(string id, string name) : base(id, name)

{

}

protected override void ProcessSalary()

{

Console.WriteLine("Architect Salary Processed");

}

}

public static class CFactory

{

public static Employee GetEmp(string id,string name,string type)

{

return type.ToUpper() switch

{

"DEV"=>new Developer(id,name),

"TES"=>new Tester(id,name),

"Arc"=>new Architect(id,name),

\_ => new Developer(id, name)

};

}

}

}

**Eg.4:(class1.cs)**

namespace NS\_IRA

{

public class CRahul

{

private int pvt = 100;

protected int prot = 200;

public int pub = 300;

internal int pvt\_inter = 400;

protected internal int prot\_inter = 500;

public void RahulFun()

{

Console.WriteLine("RahulFun");

Console.WriteLine($"pvt = {pvt}");

Console.WriteLine($"prot = {prot}");

Console.WriteLine($"pub = {pub}");

Console.WriteLine($"pvt\_inter = {pvt\_inter}");

Console.WriteLine($"prot\_inter = {prot\_inter}");

Console.WriteLine(new String('\_',40));

}

}

public class CKLRahul:CRahul

{

public void KLRahulFun()

{

Console.WriteLine("KLRahulFun");

//Console.WriteLine($"pvt = {pvt}");

Console.WriteLine($"prot = {prot}");

Console.WriteLine($"pub = {pub}");

Console.WriteLine($"pvt\_inter = {pvt\_inter}");

Console.WriteLine($"prot\_inter = {prot\_inter}");

Console.WriteLine(new String('\_', 40));

}

}

public class Home

{

CRahul rahul = new CRahul();

public void HomeFun()

{

Console.WriteLine("HomeFun");

//Console.WriteLine($"rahul.pvt = {rahul.pvt}");

//Console.WriteLine($"rahul.prot = {rahul.prot}");

Console.WriteLine($"rahul.pub = {rahul.pub}");

Console.WriteLine($"rahul.pvt\_inter = {rahul.pvt\_inter}");

Console.WriteLine($"rahul.prot\_inter = {rahul.prot\_inter}");

Console.WriteLine(new String('\_', 40));

}

}

}

**Eg.5:(Switch.cs)**

namespace NM6

{

public enum FemaleGod

{

Meenakshi=0,

Chamundeshwari=1,

Durga=2,

Kolar,

Saraswathi,

Lakshmi

}

internal class Program

{

static void Main1(string[] args)

{

Console.Write("Enter a number : ");

int choice = int.Parse(Console.ReadLine() ?? "0");

switch(choice)

{

case 0: Console.WriteLine("Apple");break;

case 1: Console.WriteLine("Orange");break;

case 2: Console.WriteLine("Pine");break;

case 3: Console.WriteLine("Mango");break;

default: Console.WriteLine("Banana");break;

}

}

static void Main2(string[] args)

{

Console.Write("Enter a number : ");

string choice = Console.ReadLine() ?? "All";

switch (choice.ToUpper())

{

case "KARNATAKA": Console.WriteLine("Chamundeshwari"); break;

case "TAMILNADU": Console.WriteLine("Meenakshi"); break;

case "KERALA": Console.WriteLine("PADMANABAN"); break;

case "ANDHRA": Console.WriteLine("VENKATACHALAPATHI"); break;

case "MAHARASHTRA": Console.WriteLine("DURGAMA"); break;

default: Console.WriteLine("RAM"); break;

}

}

static void Main3(string[] args)

{

Console.Write("Enter a number : ");

FemaleGod femgod;

try

{

femgod = (FemaleGod)Enum.Parse(typeof(FemaleGod), Console.ReadLine() ?? "Lakshmi");

}

catch (Exception)

{

Console.WriteLine("Bad Input");

return;

}

switch (femgod)

{

case FemaleGod.Meenakshi:

Console.WriteLine("Amman ");

break;

case FemaleGod.Chamundeshwari:

Console.WriteLine("Amma");

break;

case FemaleGod.Durga:

Console.WriteLine("Ma");

break;

case FemaleGod.Kolar:

Console.WriteLine("Amma");

break;

case FemaleGod.Saraswathi:

Console.WriteLine("Ma");

break;

case FemaleGod.Lakshmi:

Console.WriteLine("Ma");

break;

default:

break;

}

}

static void Main4(string[] args)

{

Console.Write("Enter a number : ");

string choice = Console.ReadLine() ?? "All";

switch (choice.ToUpper())

{

case "KARNATAKA": Console.WriteLine("Chamundeshwari"); goto case "KERALA";

case "TAMILNADU": Console.WriteLine("Meenakshi"); break;

case "KERALA": Console.WriteLine("PADMANABAN"); break;

case "ANDHRA": Console.WriteLine("VENKATACHALAPATHI"); break;

case "MAHARASHTRA": Console.WriteLine("DURGAMA"); break;

default: Console.WriteLine("RAM"); break;

}

}

static void Main5(string[] args)

{

Console.Write("The Value is : ");

//object choice = 2.3M;

//object choice = 2.3;

object choice = 2;

switch (choice)

{

case int i: Console.WriteLine($"Apple {i}"); break;

case string s: Console.WriteLine($"Orange {s}"); break;

case decimal d: Console.WriteLine($"Pine {d}"); break;

case double d1: Console.WriteLine($"Mango {d1}"); break;

default: Console.WriteLine("Banana"); break;

}

}

static void Main(string[] args)

{

Console.Write("The Value is : ");

//object choice = 2.3;

object choice = 2.3;

//object choice = 2;

//object choice = 2.3M; //M is decimal

//object choice = "rahul";

//object choice = "rahul";

switch (choice)

{

case int i when i==5: Console.WriteLine($"Apple {i}"); break;

case string s when s.Equals("sachin",StringComparison.OrdinalIgnoreCase): Console.WriteLine($"Orange {s}"); break;

case decimal d1 when d1==2.3M:

case string s1 when s1.Equals("rahul", StringComparison.OrdinalIgnoreCase):

Console.WriteLine($"Pine rahul "); break;

case double d1:

Console.WriteLine($"Mango {d1}"); break;

default: Console.WriteLine("Banana"); break;

}

}

static string GetGod(string? name)

{

FemaleGod femgod;

try

{

femgod = (FemaleGod)Enum.Parse(typeof(FemaleGod), name ?? "Lakshmi");

}

catch (Exception)

{

return "Bad Input";

}

switch (femgod)

{

case FemaleGod.Meenakshi:

return "Amman ";

case FemaleGod.Chamundeshwari:

case FemaleGod.Kolar:

return "Amma";

case FemaleGod.Durga:

case FemaleGod.Saraswathi:

case FemaleGod.Lakshmi:

return "Ma";

default:

return "No value";

}

}

static void Main7(string[] args)

{

Console.Write("Enter the god name : ");

Console.WriteLine(GetGod(Console.ReadLine()??"Lakshmi"));

}

static string GetGodFrom1(string place)

{

switch (place.ToUpper())

{

case "KARNATAKA": return "Chamundeshwari";

case "TAMILNADU": return "Meenakshi";

case "KERALA": return "PADMANABAN";

case "ANDHRA": return "VENKATACHALAPATHI";

case "MAHARASHTRA": return "DURGAMA";

default: return "RAM";

}

}

static string GetGodFrom(string place)

{

return place.ToUpper() switch

{

"KARNATAKA" => "Chamundeshwari",

"TAMILNADU" => "Meenakshi",

"KERALA" => "PADMANABAN",

"ANDHRA" => "VENKATACHALAPATHI",

"MAHARASHTRA" => "DURGAMA",

\_ => "RAM"

};

}

static void Main8(string[] args)

{

Console.Write("Enter a place : ");

Console.WriteLine(GetGodFrom(Console.ReadLine() ?? "All"));

}

static string GetAdjective(string first,string second)

{

return (first.ToUpper(),second.ToUpper()) switch

{

("SACHIN","ANJALI")=> "Cricket and Doctor",

("AZHAR", "SANGEETHA") => "Cricket and Bollywood",

("BUMRAH","SANJANA") => "Cricket and Journalist",

("CHAHAL","DANASHRI") => "Cricket and Choreographer",

("GILL","SARA") => "Cricket and Model",

\_ => "Cricket and Buttler"

};

}

static void Main9(string[] args)

{

Console.Write("Enter first name : ");

string n1 = Console.ReadLine()??"a";

Console.Write("Enter second name : ");

string n2 = Console.ReadLine()??"b";

Console.WriteLine(GetAdjective(n1,n2));

}

}

}

**Eg.6:(oops.cs)**

using System.Collections.Specialized;

namespace NM7

{

class CA

{

public CA()

{

Console.WriteLine("CA Ctor");

}

public CA(int par)

{

Console.WriteLine("CA Ctor(int)");

}

public CA(CA par)

{

Console.WriteLine("CA Copy ");

}

public void fun()

{

Console.WriteLine("CA fun");

}

~CA()

{

Console.WriteLine("CA Finalizer");

}

}

internal class CMain1

{

static void Main1(string[] args)

{

CA obj = new CA();

CA obj1 = new CA(10);

CA obj2 = new CA(obj1);

Console.WriteLine(new String('\_', 50));

obj.fun();

Console.WriteLine(new String('\_', 50));

}

}

class CB

{

public int instanceCanBeChangedAnyWhere;//instance member can be changed everywhere

public readonly int instanceCanBeChangedInCtorOnly; //can be changed in ctor only

public static int classCanBeChangedAnyWhere;

public static readonly int classCanBeChangedInStaticCtorOnly;

public const int classCannotBeChangedAtAll = 1001;

static CB()

{

classCanBeChangedAnyWhere = 1000;//can chnage

classCanBeChangedInStaticCtorOnly = 456;//can change

//e = 900;//cant change

Console.WriteLine("Static CB()");

}

public CB()

{

instanceCanBeChangedAnyWhere = 100;

instanceCanBeChangedInCtorOnly = 101;

//d = 900;

Console.WriteLine("CB Ctor");

}

public CB(int par)

{

instanceCanBeChangedAnyWhere= par;

instanceCanBeChangedInCtorOnly = par + 10;

Console.WriteLine("CB Ctor(int)");

}

public void MyFun()

{

instanceCanBeChangedAnyWhere += 10;

Console.WriteLine("CB fun");

}

~CB()

{

Console.WriteLine("CB Finalizer");

}

}

internal class CMain

{

static void Main(string[] args)

{

CB.classCanBeChangedAnyWhere = 100;//can change

CB obj1 = new CB();

CB obj2 = new CB();

CB obj3 = new CB();

obj1.instanceCanBeChangedAnyWhere = 100;

int j = obj1.instanceCanBeChangedInCtorOnly;//cant change

int k = CB.classCanBeChangedInStaticCtorOnly;//cant change

int s=CB.classCannotBeChangedAtAll;

}

}

}

**Eg.7:(libConnsumer.cs)**

using NS\_IRA;

namespace NM8

{

public class CRahulDravid : CRahul

{

public void KLRahulFun()

{

Console.WriteLine("KLRahulFun");

//Console.WriteLine($"pvt = {pvt}");

Console.WriteLine($"prot = {prot}");

Console.WriteLine($"pub = {pub}");

// Console.WriteLine($"pvt\_inter = {pvt\_inter}");

Console.WriteLine($"prot\_inter = {prot\_inter}");

Console.WriteLine(new String('\_', 40));

}

}

public class CMain

{

static void Main(string[] args)

{

CRahul rahul = new CRahul();

Console.WriteLine("HomeFun");

//Console.WriteLine($"rahul.pvt = {rahul.pvt}");

//Console.WriteLine($"rahul.prot = {rahul.prot}");

Console.WriteLine($"rahul.pub = {rahul.pub}");

// Console.WriteLine($"rahul.pvt\_inter = {rahul.pvt\_inter}");

// Console.WriteLine($"rahul.prot\_inter = {rahul.prot\_inter}");

Console.WriteLine(new String('\_', 40));

}

}

}

**Day 3**

**Eg.1:(class1.cs)**

using System;

namespace VendorLibrary

{

public delegate void MYDLG();

public delegate void MYDLG1(int x,string st);

public delegate int MYDLG2(int x,int y);

public class CVendor

{

public void DoBusiness(MYDLG dlg)

{

Console.WriteLine("Vendor Started");

dlg();//calback

Console.WriteLine("Vendor Completed");

}

public void DoBusinessNew(MYDLG1 dlg)

{

Console.WriteLine("Vendor Started");

dlg(1000,"virat and rohith left cricket t20");//calback

Console.WriteLine("Vendor Completed");

}

public void DoJob(MYDLG2 dlg)

{

Console.WriteLine("Vendor Started");

int result=dlg(1000,2000);//calback

Console.WriteLine($"Vendor Completed {result}");

}

}

}

**Eg.2:(inbuiltdelegate.cs)**

using System;

namespace VendorLibraryInBuilt

{

public delegate int MYDLG2(int x, int y);

public class CVendor

{

public void DoBusiness(Action dlg)

{

Console.WriteLine("Vendor Started1");

dlg();//calback

Console.WriteLine("Vendor Completed1");

}

public void DoBusinessNew(Action<int,string> dlg)

{

Console.WriteLine("Vendor Started1");

dlg(1000, "virat and rohith left cricket t20");//calback

Console.WriteLine("Vendor Completed1");

}

public void DoJob(Func<int,int,int> dlg)

{

Console.WriteLine("Vendor Started1");

int result = dlg(1000, 2000);//calback

Console.WriteLine($"Vendor Completed1 {result}");

}

}

}

**Eg.3:(program.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using VendorLibrary;

namespace ConsumerNs1

{

internal class Program

{

static void ClientFun()

{

Console.WriteLine("A juvinile in the line of fire is an easy target for opponents");

}

static void ClientFun1()

{

Console.WriteLine("An Extreme Famin of animals are the feast for the vultures");

}

static void ClientFun2()

{

Console.WriteLine("A rich and poor man walk for food, first one to digest it and other one to fetch it");

}

static void ClientFunNew(int val,string par)

{

Console.WriteLine($"A well Beginning is half done : vendor => [ {val} {par} ]");

}

static int ClientJob(int x,int y)

{

Console.WriteLine($"Tell me your friends, I will tell who are you : [{x},{y}] ");

return x + y + 10;

}

static void Main(string[] args)

{

MYDLG dlg = new MYDLG(ClientFun);

MYDLG dlg1 = ClientFun;

CVendor vendor = new CVendor();

vendor.DoBusiness(dlg);

Console.WriteLine(new String('\_',60));

vendor.DoBusiness(dlg1);

Console.WriteLine(new String('\_',60));

vendor.DoBusiness(ClientFun2);

Console.WriteLine(new String('\_',60));

vendor.DoBusinessNew(ClientFunNew);

Console.WriteLine(new String('\_',60));

vendor.DoJob(ClientJob);

}

}

}

**Eg.4:(programConsumer2.cs)**

using System;

using VendorLibraryInBuilt;

namespace ConsumerNs2

{

internal class Program

{

static void ClientFun()

{

Console.WriteLine("A juvinile in the line of fire is an easy target for opponents");

}

static void ClientFun1()

{

Console.WriteLine("An Extreme Famin of animals are the feast for the vultures");

}

static void ClientFun2()

{

Console.WriteLine("A rich and poor man walk for food, first one to digest it and other one to fetch it");

}

static void ClientFunNew(int val, string par)

{

Console.WriteLine($"A well Beginning is half done : vendor => [ {val} {par} ]");

}

static int ClientJob(int x, int y)

{

Console.WriteLine($"Tell me your friends, I will tell who are you : [{x},{y}] ");

return x + y + 10;

}

static void Main1(string[] args)

{

Action dlg = new Action(ClientFun);

Action dlg1 = ClientFun;

CVendor vendor = new CVendor();

vendor.DoBusiness(dlg);

Console.WriteLine(new String('\_', 60));

vendor.DoBusiness(dlg1);

Console.WriteLine(new String('\_', 60));

vendor.DoBusiness(ClientFun2);

Console.WriteLine(new String('\_', 60));

vendor.DoBusinessNew(ClientFunNew);

Console.WriteLine(new String('\_', 60));

vendor.DoJob(ClientJob);

}

static bool OddFun(int val)

{

return val % 2 == 1;

}

static void Main(string[] args)

{

Predicate<int> isOdd = OddFun;

Func<int,bool> isOdd1 = OddFun;

if(isOdd(3))

{

Console.WriteLine("3 is odd");

}

if(isOdd1(3))

{

Console.WriteLine("3 is odd");

}

}

}

}

**Eg.5:(ClousuresConsumer.cs)**

using System;

using System.Runtime.InteropServices;

namespace ConsumerNs3

{

public static class MyVendor

{

public static void VendorJob(Action act)

{

Console.WriteLine("Apple");

act();

Console.WriteLine("Orange");

Console.WriteLine(new String('\_', 60));

}

}

internal class Program

{

static void f1()

{

Console.WriteLine("f1");

}

static void f2()

{

Console.WriteLine("f2");

}

static void f3()

{

Console.WriteLine("f3");

}

static void Main(string[] args)

{

Action ac = f1;

MyVendor.VendorJob(ac);

MyVendor.VendorJob(f2);

Action ac1 = f1;

Action ac2 = f2;

Action ac3 = f3;

//multicast delegate

Action acAll = ac1 + ac2 + ac3;

MyVendor.VendorJob(acAll);

MyVendor.VendorJob(acAll-ac2);

int x = 10;

int y = 20;

Action callbackfun = delegate(){

Console.WriteLine($"Anonymous Methods x={x} and y={y}");

};

MyVendor.VendorJob(callbackfun);

Action callbackfun1 = () => { Console.WriteLine($"Lambda Expression x={x} and y={y}"); };

MyVendor.VendorJob(callbackfun1);

}

}

}

**Day 4**

**Eg.1:(class1.cs)**

using System;

using System.Configuration;

using System.Reflection;

namespace BankLib

{

public interface IDB

{

void Open();

void Close();

}

class SqlDB : IDB

{

public void Open()

{

Console.WriteLine("Open Sql DB");

}

public void Close()

{

Console.WriteLine("Close Sql DB");

}

}

class OraDB : IDB

{

public void Open()

{

Console.WriteLine("Open Ora DB");

}

public void Close()

{

Console.WriteLine("Close Ora DB");

}

}

public static class DbFactory

{

public static IDB getDb()

{

string dbLib=ConfigurationManager.AppSettings["dbLib"];

string dbType=ConfigurationManager.AppSettings["dbType"];

//reflection

Type t=Type.GetType(dbType);

IDB db=Activator.CreateInstance(t) as IDB;

//return new SqlDB();

//return new OraDB();

return db;

}

}

}

**Eg.2:(App.config)**

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />

</startup>

<appSettings>

<add key="a" value="Apple"/>

<add key="b" value="Bada Apple"/>

<add key="c" value="Chota Apple"/>

<add key="d" value="Doosra Apple"/>

<add key="e" value="Ek Aur Apple"/>

<add key="dbLib" value="BankLib"/>

<add key="dbType" value="BankLib.OraDB"/>

</appSettings>

</configuration>

**Eg.3:(001IteratorProblem.cs)**

using System;

using System.Collections;

using System.Collections.Generic;

namespace DacFeatures1

{

class CA:IEnumerable<string>

{

List<string> list = new List<string>();

int[] arr = { 10, 20, 30, 40, 50, 60, 70, 80, 90 };

public void AddData(string data)

{

list.Add(data);

}

public IEnumerable<int> Walk()

{

foreach (var item in arr)

{

yield return item;

}

}

public IEnumerator<string> GetEnumerator()

{

foreach (var item in list)

{

yield return item;

}

}

IEnumerator IEnumerable.GetEnumerator()

{

throw new NotImplementedException();

}

}

internal class CMain

{

static void Main(string[] args)

{

CA obj = new CA();

obj.AddData("sachin");

obj.AddData("saurav");

obj.AddData("rahul");

obj.AddData("sewag");

obj.AddData("vvs");

obj.AddData("yuvraj");

foreach (string item in obj)

{

Console.WriteLine(item);

}

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

foreach (var item in obj.Walk())

{

Console.Write($"{item,-10}");

}

Console.WriteLine();

}

}

}

**Eg.4:(002Featurrs.cs)**

using System;

namespace DacFeatures2

{

//rejected code (SRP violation)

class Account

{

public void SavingsJob()

{

Console.WriteLine("Open DB");

Console.WriteLine("Savings Job Done");

Console.WriteLine("Close DB");

}

public void CurrentJob()

{

Console.WriteLine("Open DB");

Console.WriteLine("Current Job Done");

Console.WriteLine("Close DB");

}

}

public class CMain

{

static void Main(string[] args)

{

Account account = new Account();

account.SavingsJob();

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

account.CurrentJob();

}

}

}

**Eg.5:(003Features.cs)**

using System;

namespace DacFeatures3

{

//rejected code(violate DIP->Dipendency Inversion Principle)

class DB

{

public void Open()

{

Console.WriteLine("Open DB");

}

public void Close()

{

Console.WriteLine("Close DB");

}

}

abstract class Account

{

DB db=new DB();

protected abstract void ActualJob();

public void DoJob()

{

db.Open();

ActualJob();

db.Close();

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

}

class Savings:Account

{

protected override void ActualJob()

{

Console.WriteLine("Savings Job Done");

}

}

class Current : Account

{

protected override void ActualJob()

{

Console.WriteLine("Current Job Done");

}

}

public class CMain

{

static void Main(string[] args)

{

Savings account = new Savings();

account.DoJob();

Current curr=new Current();

curr.DoJob();

}

}

}

**Eg.6:(004Features.cs)**

using System;

namespace DacFeatures4

{

//rejected Factory DIP

interface IDB

{

void Open();

void Close();

}

class SqlDB:IDB

{

public void Open()

{

Console.WriteLine("Open Sql DB");

}

public void Close()

{

Console.WriteLine("Close Sql DB");

}

}

class OraDB : IDB

{

public void Open()

{

Console.WriteLine("Open Ora DB");

}

public void Close()

{

Console.WriteLine("Close Ora DB");

}

}

abstract class Account

{

IDB db;

protected Account(IDB db)

{

this.db = db;

}

protected abstract void ActualJob();

public void DoJob()

{

db.Open();

ActualJob();

db.Close();

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

}

class Savings : Account

{

public Savings(IDB db):base(db)

{

}

protected override void ActualJob()

{

Console.WriteLine("Savings Job Done");

}

}

class Current : Account

{

public Current(IDB db) : base(db)

{

}

protected override void ActualJob()

{

Console.WriteLine("Current Job Done");

}

}

static class DbFactory

{

public static IDB getDb()

{

//return new SqlDB();

return new OraDB();

}

}

public class CMain

{

static void Main(string[] args)

{

IDB db = DbFactory.getDb();

Savings account = new Savings(db);

account.DoJob();

Current curr = new Current(db);

curr.DoJob();

}

}

}

**Eg.7:(005Features.cs)**

using BankLib;

using System;

using System.Configuration;

namespace DacFeatures5

{

abstract class Account

{

IDB db;

protected Account(IDB db)

{

this.db = db;

}

protected abstract void ActualJob();

public void DoJob()

{

db.Open();

ActualJob();

db.Close();

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

}

class Savings : Account

{

public Savings(IDB db) : base(db)

{

}

protected override void ActualJob()

{

Console.WriteLine("Savings Job Done");

}

}

class Current : Account

{

public Current(IDB db) : base(db)

{

}

protected override void ActualJob()

{

Console.WriteLine("Current Job Done");

}

}

public class CMain

{

static void Main(string[] args)

{

IDB db = DbFactory.getDb();

Savings account = new Savings(db);

account.DoJob();

Current curr = new Current(db);

curr.DoJob();

}

static void Main2(string[] args)

{

Console.WriteLine(ConfigurationManager.AppSettings["a"]);

Console.WriteLine(ConfigurationManager.AppSettings["b"]);

Console.WriteLine(ConfigurationManager.AppSettings["c"]);

Console.WriteLine(ConfigurationManager.AppSettings["d"]);

Console.WriteLine(ConfigurationManager.AppSettings["e"]);

Console.WriteLine(ConfigurationManager.AppSettings["dbLib"]);

Console.WriteLine(ConfigurationManager.AppSettings["dbType"]);

}

}

}

**Day4\_2**

**DesignPatternSoln**

**Eg.1:(Program.cs)**

using System;

using System.Collections.Generic;

//shubam (commad pattern [undo])

namespace DesignPattern

{

class Account

{

int amount = 0;

public Account(int amount)

{

this.amount = amount;

}

public void Debit(int amt)

{

this.amount-=amt;

}

public void Credit(int amt)

{

this.amount += amt;

}

public int Balance()

{

return amount;

}

}

interface ICommand

{

int Amount { get; }

void Execute(Account acc);

}

class DebitCommand : ICommand

{

public DebitCommand(int amt)

{

Amount = amt;

}

public int Amount { get; }

public void Execute(Account acc)

{

acc.Debit(Amount);

}

}

class CreditCommand : ICommand

{

public CreditCommand(int amt)

{

Amount = amt;

}

public int Amount { get; }

public void Execute(Account acc)

{

acc.Credit(Amount);

}

}

class CommadnRepo

{

Stack<ICommand> commands = new Stack<ICommand>();

public void Add(ICommand cmd)

{

commands.Push(cmd);

}

public ICommand GetCommand()

{

return commands.Pop();

}

}

class TransHelper

{

Account account;

CommadnRepo repo = new CommadnRepo();

public TransHelper(Account account)

{

this.account = account;

}

public void Withdraw(int amt)

{

repo.Add(new CreditCommand(amt));

; account.Debit(amt);

}

public void Deposit(int amt)

{

repo.Add(new DebitCommand(amt));

account.Credit(amt);

}

public int Balance

{

get { return account.Balance(); }

}

public void Undo()

{

ICommand command = repo.GetCommand();

command.Execute(account);

}

}

internal class Program

{

static void Main(string[] args)

{

Account account = new Account(5000);

TransHelper transHelper = new TransHelper(account);

transHelper.Deposit(1000);

transHelper.Deposit(2000);

transHelper.Withdraw(1000);

transHelper.Withdraw(1000);

transHelper.Undo();

transHelper.Undo();

transHelper.Undo();

Console.WriteLine(transHelper.Balance);

}

}

}

**TestD1.ShoppingCart**

**Eg.1:(Program.cs)**

using Ninject;

using System;

using System.Collections.Generic;

namespace TestDI1

{

interface ICalculator

{

decimal Sum(IEnumerable<Product> products);

}

class Product

{

public string Name { get; set; }

public decimal Price { get; set; }

}

class ShoppingCart

{

ICalculator calculator;

public ShoppingCart(ICalculator calculator)

{

this.calculator = calculator;

}

List<Product> products=new List<Product>();

public void AddToCart(string name,decimal price)

{

products.Add(new Product() { Name = name, Price = price });

}

public decimal Total()

{

return calculator.Sum(products);

}

public void Display()

{

Array.ForEach(products.ToArray(), p => Console.WriteLine($"{p.Name,-15}{p.Price,10}"));

}

}

class DacCalculator: ICalculator

{

public DacCalculator()

{

Console.WriteLine("DacCalculator");

}

public decimal Sum(IEnumerable<Product> products)

{

decimal cost = 0.0M;

foreach (Product product in products)

{

cost+=product.Price;

}

return cost;

}

}

class CDacCalculator : ICalculator

{

IDiscount discount;

public CDacCalculator(IDiscount discount)

{

this.discount = discount;

Console.WriteLine("CDacCalculator");

}

public decimal Sum(IEnumerable<Product> products)

{

decimal cost = 0.0M;

foreach (Product product in products)

{

cost += product.Price\*1.1M-discount.discount();

}

return cost;

}

}

interface IDiscount

{

decimal discount();

}

class Christmas: IDiscount

{

public Christmas()

{

Console.WriteLine("Christmas");

}

public decimal discount()

{

return 100M;

}

}

class Diwalitmas : IDiscount

{

public Diwalitmas()

{

Console.WriteLine("Diwalitmas");

}

public decimal discount()

{

return 200M;

}

}

internal class Program

{

static void AddCartItems(ShoppingCart cart)

{

cart.AddToCart("gloves", 234.5M);

cart.AddToCart("pads", 500M);

cart.AddToCart("guard", 120.23M);

cart.AddToCart("helmet", 234.56M);

cart.AddToCart("shoes", 345.5M);

}

static void Main1(string[] args)

{

DacCalculator calculator = new DacCalculator();

// ShoppingCart cart = new ShoppingCart(calculator);

//CDacCalculator calculator = new CDacCalculator();

ShoppingCart cart = new ShoppingCart(calculator);

AddCartItems(cart);

cart.Display();

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine($"{"Total",-15}{cart.Total(),10}");

}

static IKernel getKetnel()

{

IKernel kernel = new StandardKernel();

// kernel.Bind<ICalculator>().To<DacCalculator>();

//kernel.Bind<IDiscount>().To<Christmas>();

kernel.Bind<IDiscount>().To<Diwalitmas>();

kernel.Bind<ICalculator>().To<CDacCalculator>();

return kernel;

}

static void Main(string[] args)

{

IKernel kernel=getKetnel();

ShoppingCart cart = kernel.Get<ShoppingCart>();

AddCartItems(cart);

cart.Display();

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine($"{"Total",-15}{cart.Total(),10}");

}

}

}

**Eg.2:(TestDi2.cs)**

using Ninject;

using System;

using System.Collections.Generic;

namespace TestDI2

{

class Product

{

public string Name { get; set; }

public decimal Price { get; set; }

}

interface IData

{

List<Product> Products { get; }

}

class DataAccess

{

IData data;

public DataAccess(IData data)

{

this.data = data;

}

public void Display()

{

Array.ForEach(data.Products.ToArray(), p => Console.WriteLine($"{p.Name,-15}{p.Price,10}"));

}

}

class Mydata:IData

{

List <Product> prods=new List<Product>();

public List<Product> Products { get { return prods; } }

public Mydata()

{

prods.Add(new Product() {Name="cat",Price=123.45M });

prods.Add(new Product() {Name="dog",Price=143.45M });

prods.Add(new Product() {Name="tiger",Price=103.45M });

prods.Add(new Product() {Name="cheetah",Price=13.45M });

prods.Add(new Product() {Name="kohli",Price=12.45M });

}

}

class CA : IDisposable

{

public CA()

{

Console.WriteLine("CA ctor");

}

public void Dispose()

{

Console.WriteLine("CA Disposed");

}

}

class Testdata : IData

{

List<Product> prods = new List<Product>();

public List<Product> Products { get { return prods; } }

public Testdata(IDisposable dis)

{

prods.Add(new Product() { Name = "cat1111", Price = 123.45M });

prods.Add(new Product() { Name = "dog1111", Price = 143.45M });

prods.Add(new Product() { Name = "tiger1111", Price = 103.45M });

prods.Add(new Product() { Name = "cheetah1111", Price = 13.45M });

prods.Add(new Product() { Name = "kohli1111", Price = 12.45M });

dis.Dispose();

}

}

static class CKernelFactory

{

public static IKernel GetKernel()

{

StandardKernel kernel = new StandardKernel();

// kernel.Bind<IData>().To<Mydata>();

kernel.Bind<IData>().To<Testdata>();

kernel.Bind<IDisposable>().To<CA>();

return kernel;

}

}

class CMain

{

static void Main(string[] args)

{

IKernel kernel = CKernelFactory.GetKernel();

DataAccess da = kernel.Get<DataAccess>();

da.Display();

}

}

}

**Eg.3:(Events3.cs)**

using System;

namespace Events3

{

delegate void Handler(string eveType);

class Button

{

public event Handler Click;

public void OnClick()

{

if (Click != null)

Click("clicked");

}

}

class CMain

{

static void Main(string[] args)

{

Button button = new Button();

button.Click += (str) => Console.WriteLine("I am Clicked1");

button.Click += str => Console.WriteLine("I am Clicked2");

button.Click += str => Console.WriteLine("I am Clicked3");

button.OnClick();

}

}

}

**Day5**

**ConsoleAppData.Transaction**

**Eg.1:(Class1.cs)**

using System.Text;

using System.Threading.Tasks;

namespace DataAccessContract

{

public class EmpOrm

{

public int E\_Id { get; set; }

public string E\_Name { get; set; }

public int E\_Did { get; set; }

}

public interface IData

{

EmpOrm[] GetAllEmployee();

EmpOrm GetAnEmployee(int id);

bool AddEmployee(EmpOrm empOrm);

bool DeleteEmployee(int id);

bool ModifyEmployee(EmpOrm empOrm);

}

}

**Eg.2:(001DataAcess.cs)**

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleAppData

{

internal class Program

{

static void Main1(string[] args)

{

//step 1 create a connection

SqlConnection cnn = new SqlConnection();

cnn.ConnectionString = @"Data Source=.\sqlexpress;Initial Catalog=MonishDb;Integrated Security=true";

//step 2 create a command

SqlCommand cmd = new SqlCommand();

cmd.CommandText = "select \* from Employee";

//step 3 associate command and connection

cmd.Connection=cnn;

//step 5 open connection

cnn.Open();

//step 4 create a data reader

SqlDataReader reader=cmd.ExecuteReader();

//step iterate through reader

while(reader.Read())

{

Console.WriteLine($"{reader[0],-15}{reader[1],-15}{reader[2],-15}");

}

reader.Close();

cnn.Close();

}

static void Main2(string[] args)

{

SqlConnection cnn = new SqlConnection(@"Data Source=.\sqlexpress;Initial Catalog=MonishDb;Integrated Security=true");

SqlCommand cmd = new SqlCommand("select \* from Employee", cnn);

cnn.Open();

SqlDataReader reader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

while (reader.Read())

{

Console.WriteLine($"{reader[0],-15}{reader[1],-15}{reader[2],-15}");

}

reader.Close();

}

static void Main3(string[] args)

{

SqlConnection cnn = new SqlConnection(ConfigurationManager.ConnectionStrings["apple"].ConnectionString);

SqlCommand cmd = new SqlCommand("select \* from Employee", cnn);

cnn.Open();

SqlDataReader reader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

while (reader.Read())

{

Console.WriteLine($"{reader[0],-15}{reader[1],-15}{reader[2],-15}");

}

reader.Close();

}

static void Main4(string[] args)

{

SqlConnection cnn = new SqlConnection(ConfigurationManager.ConnectionStrings["apple"].ConnectionString);

SqlCommand cmd = new SqlCommand("insert into Employee values(109,'kapildev',10)", cnn);

cnn.Open();

int recordsAffeted=cmd.ExecuteNonQuery();

cnn.Close();

Main3(null);

}

static void Main(string[] args)

{

SqlConnection cnn = new SqlConnection(ConfigurationManager.ConnectionStrings["apple"].ConnectionString);

SqlCommand cmd = new SqlCommand("select count(\*) from Employee", cnn);

cnn.Open();

int count = (int)cmd.ExecuteScalar();

cnn.Close();

Console.WriteLine($"Count = {count}");

}

}

}

**Eg.3:(002TestDataAccess)**

using ConsoleAppData.Model;

using DataAccessContract;

using Ninject;

using System;

namespace ConsoleAppData

{

class Transaction

{

IData data;

public Transaction(IData data)

{

this.data = data;

}

public void DisplayEmployee()

{

Array.ForEach(data.GetAllEmployee(), emp => Console.WriteLine($"{emp.E\_Id,-15}{emp.E\_Name,-15}{emp.E\_Did,-15}"));

}

public void Display(int id)

{

EmpOrm emp=data.GetAnEmployee(id);

if (emp != null)

Console.WriteLine($"{emp.E\_Id,-15}{emp.E\_Name,-15}{emp.E\_Did,-15}");

}

public void AddNewEmployee(int id,string name,int did)

{

bool res= data.AddEmployee(new EmpOrm { E\_Id = id, E\_Name = name, E\_Did = did });

if(res)

Console.WriteLine("Added Successfully");

else

Console.WriteLine("Not Added ");

}

}

public class CMain

{

static IKernel GetKernel()

{

StandardKernel kernel = new StandardKernel();

//kernel.Bind<IData>().To<ActualRepository>();

kernel.Bind<IData>().To<FakeRepository>();

return kernel;

}

static void Main(string[] args)

{

//Transaction trans=new Transaction(new FakeRepository());

// Transaction trans=new Transaction(new ActualRepository());

IKernel kernel = GetKernel();

Transaction trans = kernel.Get<Transaction>();

trans.DisplayEmployee();

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

trans.Display(113);

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

// trans.AddNewEmployee(117, "apricot", 10);

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

trans.DisplayEmployee();

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

}

}

**Eg.4:(ActualRepository.cs)**

using DataAccessContract;

using System;

using System.Collections.Generic;

using System.ComponentModel.Design;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

namespace ConsoleAppData.Model

{

public class ActualRepository : IData

{

SqlConnection cnn = new SqlConnection(ConfigurationManager.ConnectionStrings["apple"].ConnectionString);

SqlCommand cmd ;

public ActualRepository()

{

cmd = new SqlCommand("", cnn);

}

public bool AddEmployee(EmpOrm empOrm)

{

cmd.CommandText = $"insert into Employee values({empOrm.E\_Id},'{empOrm.E\_Name}',{empOrm.E\_Did})";

cnn.Open();

int recordsAffeted = cmd.ExecuteNonQuery();

cnn.Close();

return recordsAffeted>0;

}

public bool DeleteEmployee(int id)

{

cmd.CommandText = $"delete from Employee where EmpId={id}";

cnn.Open();

int recordsAffeted = cmd.ExecuteNonQuery();

cnn.Close();

return recordsAffeted > 0;

}

public EmpOrm[] GetAllEmployee()

{

List<EmpOrm> employees = new List<EmpOrm>();

cmd.CommandText = "select \* from Employee";

cnn.Open();

SqlDataReader reader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

while (reader.Read())

{

employees.Add(new EmpOrm { E\_Id= (int)reader[0], E\_Name= reader[1].ToString(), E\_Did=(int)reader[2] });

}

reader.Close();

return employees.ToArray();

}

public EmpOrm GetAnEmployee(int id)

{

List<EmpOrm> employees = new List<EmpOrm>();

cmd.CommandText = $"select \* from Employee where EmpId={id}";

cnn.Open();

SqlDataReader reader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

while (reader.Read())

{

employees.Add(new EmpOrm { E\_Id = (int)reader[0], E\_Name = reader[1].ToString(), E\_Did = (int)reader[2] });

}

reader.Close();

return (employees.Count > 0) ? employees[0] : null;

}

public bool ModifyEmployee(EmpOrm empOrm)

{

//??

return true;

}

}

}

**Eg.5:(FakeRepository)**

using DataAccessContract;

using System.Collections.Generic;

using System.ComponentModel.Design;

using System.Linq;

namespace ConsoleAppData.Model

{

public class FakeRepository : IData

{

static List<EmpOrm> employees=new List<EmpOrm>();

static FakeRepository()

{

employees.Add(new EmpOrm { E\_Id = 111, E\_Name = "apple", E\_Did = 1223 });

employees.Add(new EmpOrm { E\_Id = 112, E\_Name = "orange", E\_Did = 12323 });

employees.Add(new EmpOrm { E\_Id = 113, E\_Name = "pine", E\_Did = 1223 });

employees.Add(new EmpOrm { E\_Id = 114, E\_Name = "jack", E\_Did = 3333 });

employees.Add(new EmpOrm { E\_Id = 115, E\_Name = "mango", E\_Did = 3333 });

}

public bool AddEmployee(EmpOrm empOrm)

{

employees.Add(empOrm);

return true;

}

public bool DeleteEmployee(int id)

{

employees =new List<EmpOrm>( employees.Where(e => e.E\_Id != id));

return true;

}

public EmpOrm[] GetAllEmployee()

{

return employees.ToArray();

}

public EmpOrm GetAnEmployee(int id)

{

return (from emp in employees where emp.E\_Id==id select emp).First();

}

public bool ModifyEmployee(EmpOrm empOrm)

{

EmpOrm emp = employees.Where(e => e.E\_Id != empOrm.E\_Id).First();

emp.E\_Name = empOrm.E\_Name;

emp.E\_Did = empOrm.E\_Did;

return true;

}

}

}

**ConsoleAppGC**

**Eg.1:(GCCollect.cs)**

using System;

using System.Threading;

namespace ConsoleAppGC

{

class CA

{

public CA()

{

Console.WriteLine("CA Ctor");

}

~CA()

{

Console.WriteLine("CA Finalizer");

}

}

class CC

{

public CC()

{

Console.WriteLine("CC Ctor");

}

}

class CB

{

public CB()

{

Console.WriteLine("CB Ctor");

}

~CB()

{

for (int i = 0; i < 5; i++)

{

Console.WriteLine("CB Finalizer");

Thread.Sleep(1000);

}

}

}

//freachable thread

internal class Program

{

static void CreateCA()

{

CA obj = new CA();

}

static void Main1(string[] args)

{

CA obj1 = new CA();

CreateCA();

Console.WriteLine($"obj1 in gen : {GC.GetGeneration(obj1)}");

GC.Collect(0);

//Thread.Sleep(1000);

Console.WriteLine($"obj1 in gen : {GC.GetGeneration(obj1)}");

GC.Collect(1);

Console.WriteLine($"obj1 in gen : {GC.GetGeneration(obj1)}");

}

static void CreateCB()

{

CB obj = new CB();

}

static void Main2(string[] args)

{

CreateCB();

GC.Collect(0);

for (int i = 0; i < 5; i++)

{

Console.WriteLine("\t\tApple");

Thread.Sleep(1000);

}

}

static void Main(string[] args)

{

CreateCB();

GC.Collect(0);

GC.WaitForPendingFinalizers();

GC.Collect(0);

for (int i = 0; i < 5; i++)

{

Console.WriteLine("\t\tApple");

Thread.Sleep(1000);

}

}

static void CreateCC()

{

CC obj = new CC();

}

static void Main4(string[] args)

{

CA obj1 = new CA();

CreateCA();

CC obj2 = new CC();

CreateCC();

GC.Collect(0);

}

}

}

**Eg.2:(Concepts.cs)**

using System;

namespace ConsoleApp3

{

//value types -> stack [primitive,struct,enums]

//reference -> heap [arrays,class,delegates]

//generics

class CA<T>

{

T i = default(T);

public CA(T x)

{

Console.WriteLine($"CA for {x.GetType().Name}");

i = x;

}

public void Display()

{

Console.WriteLine($"Display i [{typeof(T).Name}]");

}

public override string ToString() => $"toString i={i}";

}

class CB<T>

// class CB<T> where T : struct //allow only value types

// class CB<T> where T : class //allow only reference types

// class CB<T> where T : class,new() //allow only class with default ctor

// class CB<T> where T : class,IDisposable //allow only class implementing IDisposable

{

}

struct CAT

{

}

class DOG

{

public DOG(int v)

{

}

}

class Hen : IDisposable

{

public void Dispose()

{

}

}

public class CMain4

{

static void Main1(string[] args)

{

CA<int> obj1=new CA<int>(100);

CA<double> obj2 = new CA<double>(3.141);

CA<object> obj3 = new CA<object>(new object());

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

obj1.Display();

obj2.Display();

obj3.Display();

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine(obj1);

Console.WriteLine(obj2);

Console.WriteLine(obj3);

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

static void Main(string[] args)

{

CB<int> obj1 = new CB<int>();

CB<CAT> obj4 = new CB<CAT>();

CB<CA<double>> obj2 = new CB<CA<double>>();

CB<int[]> obj3 = new CB<int[]>();

CB<DOG> obj5 = new CB<DOG>();

CB<Hen> obj6 = new CB<Hen>();

}

}

}

**Eg.3:(IndexCS.cs)**

using System;

namespace ConsoleApp4

{

class CA

{

int[] arr = new int[] { 1, 2, 3, 45, 6, 7 };

public int this[int index]

{

get { return arr[index]; }

set { arr[index] = value; }

}

public int this[string index]

{

get { return arr[int.Parse(index)]; }

set { arr[int.Parse(index)] = value; }

}

public void Display()

{

Array.ForEach(arr, val=>Console.WriteLine(val));

}

}

class CComplex

{

double \_real = 0.0;

double \_imag = 0.0;

public CComplex(double real=0.0, double imag=0.0)

{

\_real = real;

\_imag = imag;

}

public static CComplex operator +(CComplex c1, CComplex c2) => new CComplex(c1.\_real+c2.\_real,c1.\_imag+c2.\_imag);

//public static implicit operator double(CComplex c) => c.\_real + c.\_imag;

public static explicit operator double(CComplex c) => c.\_real + c.\_imag;

public override string ToString() => $"{\_real}{(\_imag >= 0.0 ? "+" : "")}{\_imag}i";

}

class CMain

{

static void Main1(string[] args)

{

CA obj=new CA();

obj.Display();

obj[3] = 4;

obj["2"] = 999;

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

obj.Display();

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

int k = obj[3];

Console.WriteLine($"k={k}");

}

static void Main(string[] args)

{

CComplex obj=new CComplex();

CComplex obj1=new CComplex(1.34,-4.5);

CComplex obj2=new CComplex(1.4,4.5);

foreach (var item in new[] { obj,obj1,obj2})

{

Console.WriteLine(item);

}

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

CComplex obj3 = obj1 + obj2;

Console.WriteLine(obj3);

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

//double d = obj3;

double d = (double)obj3;

Console.WriteLine($"d={d}");

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

}

}

**ConsoleAppThread2**

**Eg.1:(CMain@.cs)**

using System;

using System.Threading;

//race condition

namespace ConsoleAppThread2

{

class CA

{

int x=0;

int y=0;

public void AddFun()

{

for (x = 1; x <= 5; x++)

{

Console.WriteLine($"AddFun x : {x}");

Thread.Sleep(1000);

}

for (y = 1; y <= 10; y++)

{

Console.WriteLine($"AddFun y : {y}");

Thread.Sleep(1000);

}

}

public void SubFun()

{

for (y = -1; y > -10; y--)

{

Console.WriteLine($"\t\tSubFun y : {y}");

Thread.Sleep(1000);

}

for (x = -1; x > -5; x--)

{

Console.WriteLine($"\t\tSubFun x : {x}");

Thread.Sleep(1000);

}

}

}

class CMain2

{

static void Main(string[] args)

{

CA obj = new CA();

ThreadStart ts1=new ThreadStart(obj.AddFun);

ThreadStart ts2=new ThreadStart(obj.SubFun);

Thread t1 = new Thread(ts1);

Thread t2 = new Thread(ts2);

t1.Start();

t2.Start();

t1.Join();

t2.Join();

}

}

}

**Eg.2:(CMain3.cs)**

using System;

using System.Threading;

namespace ConsoleAppThread3

{

class CA

{

static object mylockobj=new object();

int x = 0;

int y = 0;

public void AddFun()

{

for (x = 1; x <= 5; x++)

{

Console.WriteLine($"AddFun x : {x}");

Thread.Sleep(1000);

}

lock (mylockobj)

{ //Monitor.Enter(mylockobj);

for (y = 1; y <= 10; y++)

{

Console.WriteLine($"AddFun y : {y}");

Thread.Sleep(1000);

}

}// Monitor.Exit(mylockobj);

}

public void SubFun()

{

Monitor.Enter(mylockobj);

for (y = -1; y > -10; y--)

{

Console.WriteLine($"\t\tSubFun y : {y}");

Thread.Sleep(1000);

}

Monitor.Exit(mylockobj);

for (x = -1; x > -5; x--)

{

Console.WriteLine($"\t\tSubFun x : {x}");

Thread.Sleep(1000);

}

}

}

class CMain3

{

static void Main(string[] args)

{

CA obj = new CA();

ThreadStart ts1 = new ThreadStart(obj.AddFun);

ThreadStart ts2 = new ThreadStart(obj.SubFun);

Thread t1 = new Thread(ts1);

Thread t2 = new Thread(ts2);

t1.Start();

t2.Start();

t1.Join();

t2.Join();

}

}

}

**Eg.3:(CMain4.cs)**

using System;

using System.Threading;

namespace ConsoleAppThread4

{

class CA

{

Mutex mute = new Mutex(false, "MyMutex");

int x = 0;

int y = 0;

public void AddFun()

{

for (x = 1; x <= 5; x++)

{

Console.WriteLine($"AddFun x : {x}");

Thread.Sleep(1000);

}

mute.WaitOne();

for (y = 1; y <= 10; y++)

{

Console.WriteLine($"AddFun y : {y}");

Thread.Sleep(1000);

}

mute.ReleaseMutex();

}

public void SubFun()

{

mute.WaitOne();

for (y = -1; y > -10; y--)

{

Console.WriteLine($"\t\tSubFun y : {y}");

Thread.Sleep(1000);

}

mute.ReleaseMutex();

for (x = -1; x > -5; x--)

{

Console.WriteLine($"\t\tSubFun x : {x}");

Thread.Sleep(1000);

}

}

}

class CMain4

{

static void Main(string[] args)

{

CA obj = new CA();

ThreadStart ts1 = new ThreadStart(obj.AddFun);

ThreadStart ts2 = new ThreadStart(obj.SubFun);

Thread t1 = new Thread(ts1);

Thread t2 = new Thread(ts2);

t1.Start();

t2.Start();

t1.Join();

t2.Join();

}

}

}

/\*

Manualresetevent

Autoresetevent

Barrier

CountdownEvent

\*/