31927 32998:
Application Development with .NET

Week-3 Lecture

C# Programming Basics

Part-2

Outline

- Arrays in C#
- User defined methods
- Programming Paradigms
- Object Oriented Programming basics
- Class and Methods
- Constructors

Arrays in C#

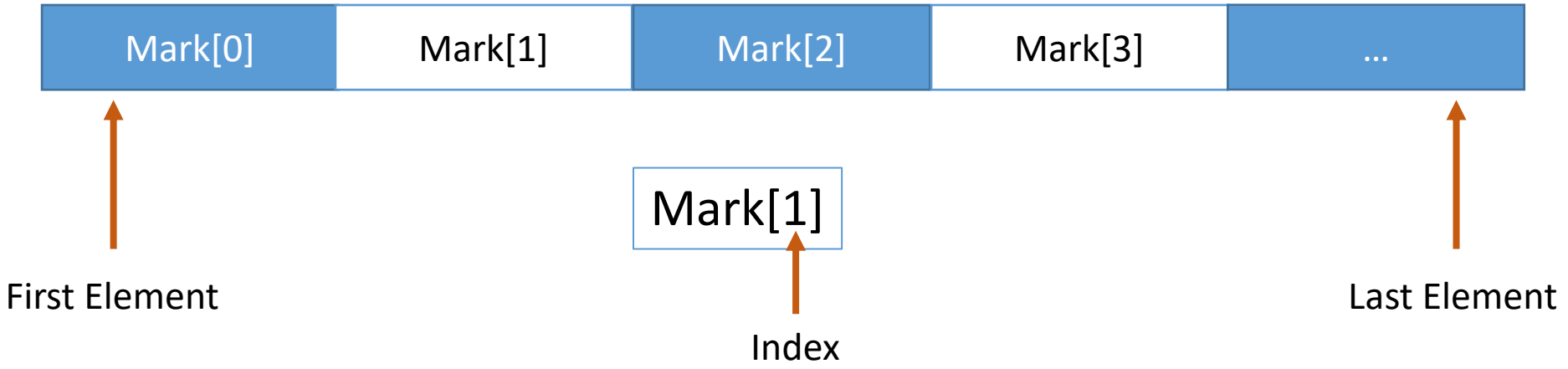
- An Array is a collection of data/elements of same type
- Arrays are of fixed-size and stored at contiguous/sequential memory locations.
- Example: instead of declaring 1000 variables to store marks of 1000 students, such as: mark1, mark2, …, mark1000,

  Declare one array variable of type float/int such as mark and use

  mark[0], mark[1], …, mark[999] to represent individual variables
- A particular element in an array is accessed by an index
- Array is a reference type!

Reference: https://www.tutorialspoint.com/csharp/csharp_arrays.htm

Arrays in C#



- Declaring Arrays:

  <datatype>[] arrayVariableName;

- Example:

  Float[] mark;

Reference: https://www.tutorialspoint.com/csharp/csharp_arrays.htm

Arrays in C#

1. Initializing an Array:

```
double[] marks = new double[1000];
```

new keyword is used to create an instance of type array.

2. Assigning values to an Array:

a. Marks[0] = 98.0;

b. double[] marks = {98.0, 79.0, 65.0};

c. double[] marks = new double[] {98.0, 79.0, 65.0};

e. double[] marks = new double[3] {98.0, 79.0, 65.0};

Reference: https://www.tutorialspoint.com/csharp/csharp_arrays.htm

Arrays in C#

3. The Length property :

The size of an array can be found in the Length property

Example:

```
double[] marks = {98.0, 79.0, 65.0};
int arraySize = marks.Length;
Console.Writeline("The array size is : {0}", arraySize);
```

Output: The array size is : 3

Reference: https://www.tutorialspoint.com/csharp/csharp_arrays.htm

Arrays in C#

4. Accessing Array elements:

```
double myMarks = marks[2];
```

Output

```
Enter the mark for Student-1: 95
Enter the mark for Student-2: 96
Enter the mark for Student-3: 86
Enter the mark for Student-4: 89
Enter the mark for Student-5: 79
Enter the mark for Student-6: 91
Enter the mark for Student-7: 82
Enter the mark for Student-8: 94
Enter the mark for Student-9: 76
Enter the mark for Student-10: 99

There are 10 elements in the marks array

You have entered the folowing values:
Mark 1 : 95
Mark 2 : 96
Mark 3 : 86
Mark 4 : 89
Mark 5 : 79
Mark 6 : 91
Mark 7 : 82
Mark 8 : 94
Mark 9 : 76
Mark 10 : 99
```

Example:

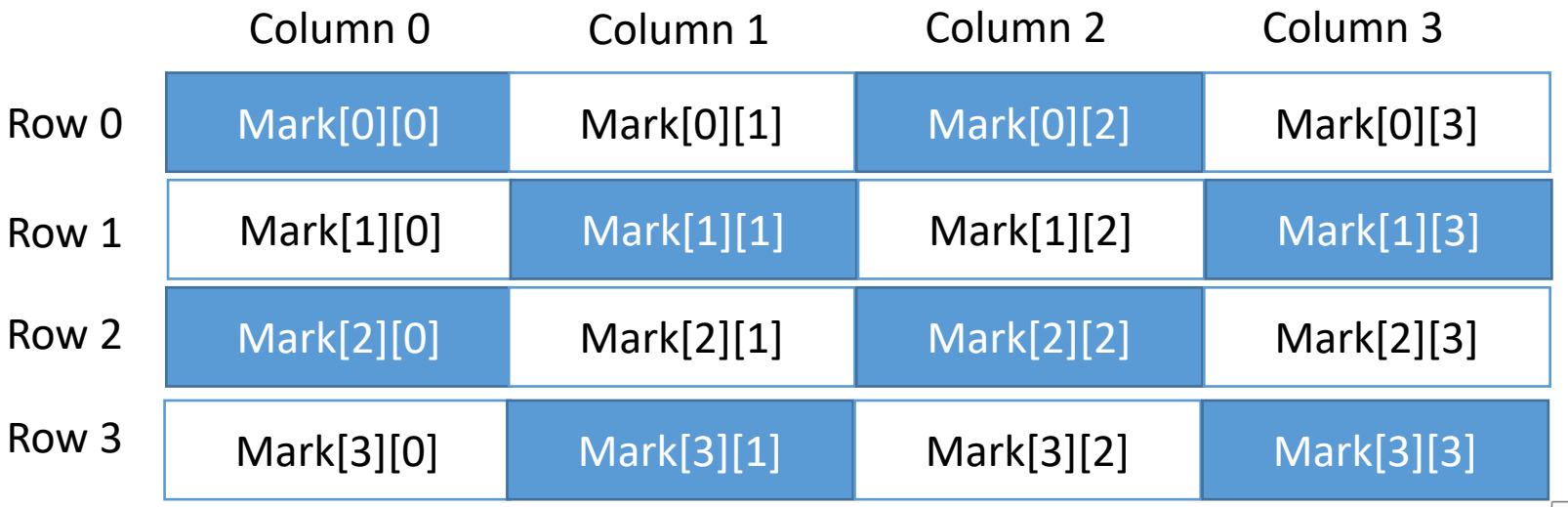
```
using System;

//Program demonstrates the use of arrays
namespace Week3Programs
{
    0 references
    class ArrayDemo
    {
        0 references
        static void Main(string[] args)
        {
            // Declare an array of size 10
            // to store marks of 10 students
            double[] marks = new double[10];
            string userInput;
            // Initialize the marks array with user inputs
            for(int loopControlVar = 0; loopControlVar <10; loopControlVar++)
            {
                // Display message to the user
                Console.Write("Enter the mark for Student-{0}: ", loopControlVar + 1);
                // Read user input and store in a string variable
                userInput = Console.ReadLine();
                // Convert the user input from string to Double
                marks[loopControlVar] = Convert.ToDouble(userInput);
            }
            // Compute array lenght and display
            int arraySize = marks.Length;
            Console.WriteLine("\nThere are {0} elements in the marks array", arraySize);
            Console.WriteLine("\nYou have entered the folowing values:");

            // Display the values provided by the user
            for (int loopControlVar = 0; loopControlVar <10; loopControlVar++)
            {
                // Display the values
                Console.WriteLine("Mark {0} : {1}", loopControlVar + 1, marks[loopControlVar]);
            }
            //
            Console.ReadKey();
        }
    }
}
```

Reference: https://www.tutorialspoint.com/csharp/csharp_arrays.htm

Arrays in C#

5. Two-dimensional arrays:

- 2D arrays are simple form of multi-dimensional array.
- A 2D array is like a table which has rows and columns



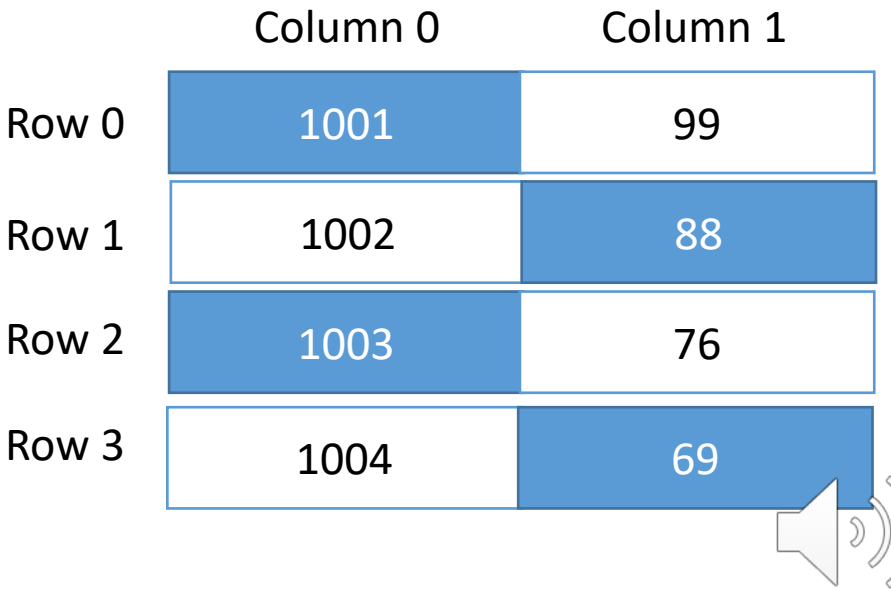
Reference: https://www.tutorialspoint.com/csharp/csharp_arrays.htm

Arrays in C#

5. Two-dimensional arrays:

- **Initialize 2D array:**

```
int[,] studentInfo = new double [4, 2] {
    { 1001, 99}, // initializes 1st row elements
    { 1002, 88}, // initializes 2nd row elements
    { 1003, 76}, // initializes 3rd  row elements
    { 1004, 69}, // initializes 4th row elements
};
```



Reference: https://www.tutorialspoint.com/csharp/csharp_arrays.htm

Arrays in C#

Two-dimensional arrays Example:

```
using System;
// Programs demonstrate 2D array

namespace Week3Program
{
    0 references
    class TwoDArrayDemo
    {
        0 references
        static void Main(string[] args)
        {
            // Declare a 2D array to store roll number and marks
            // Assume the roll number and marks are integer
            int[,] studentInfo = new int[3, 2];
            string userInput;
            // Declare a string array to store student's name
            string[] studentName = new string[3];

            //Initialize the arrays from user inputs
            for(int outerLoopRow = 0; outerLoopRow < 3; outerLoopRow++)
            {
                // Read the Student's Name
                Console.Write("Enter the student's name: ");
                studentName[outerLoopRow] = Console.ReadLine();
                for(int innerLoopCol = 0; innerLoopCol<2; innerLoopCol++)
                {
                    if (innerLoopCol == 0)
                        Console.Write("Enter the student's roll number: ");
                    else if (innerLoopCol == 1)
                        Console.Write("Enter the student's mark: ");

                    // Read the student's roll and mark from user
                    userInput = Console.ReadLine();
                    studentInfo[outerLoopRow, innerLoopCol] = Convert.ToInt32(userInput);
                }
            }
            // Display students details
            for (int outerLoopRow = 0; outerLoopRow < 3; outerLoopRow++)
            {
                // Display the Student's Name
                Console.WriteLine("\nYou have entered the following details:");
                Console.WriteLine("Student Name : {0}", studentName[outerLoopRow]);
                for (int innerLoopCol = 0; innerLoopCol < 2; innerLoopCol++)
                {
                    if (innerLoopCol == 0)
                        // Display roll number
                        Console.WriteLine("Roll number: {0}", studentInfo[outerLoopRow, innerLoopCol]);
                    else if (innerLoopCol == 1)
                        // Disply Mark
                        Console.WriteLine("Mark Obtained: {0} \n", studentInfo[outerLoopRow, innerLoopCol]);
                }
            }
            // Read a key from the user
            Console.ReadKey();
        }
    }
}
```

Output:

```
Enter the student's name: Nabin Sharma
Enter the student's roll number: 1001
Enter the student's mark: 95
Enter the student's name: Sam
Enter the student's roll number: 1002
Enter the student's mark: 92
Enter the student's name: Lily
Enter the student's roll number: 1003
Enter the student's mark: 91

You have entered the following details:
Student Name : Nabin Sharma
Roll number: 1001
Mark Obtained: 95


You have entered the following details:
Student Name : Sam
Roll number: 1002
Mark Obtained: 92


You have entered the following detai
Student Name : Lily
Roll number: 1003
Mark Obtained: 91
```

User Defined Methods in C#

- Method is a group of statements packaged within a scope, that perform a specific task.

- A C# program has at least one class and one method – Main()

Reference: https://www.tutorialspoint.com/csharp/csharp_methods.htm

User Defined Methods in C#

- Syntax to define a method:

<Access_Specifier> <Return_type> <**Method_name**> (Parameter list)

{

// Method body

}

Where:

1. <Access_Specifier>: Determines the visibility of the method.
2. <Return_type>: Data type of the value returned by the method. Such as, int, double, String, etc.
   If a method doesn't return any value, the return type should be **void**.
3. **<Method_name>:** A case sensitive unique identifier

Parameter List: Values/data passed to the method (optional)

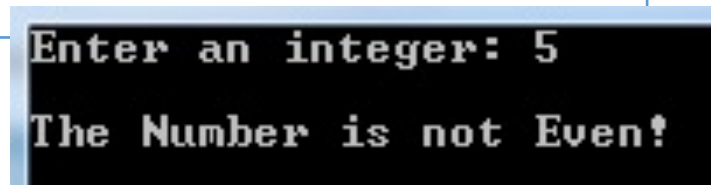
Reference: https://www.tutorialspoint.com/csharp/csharp_methods.htm

User Defined Methods in C#

- Example:

```
// Create a new method checks whether a number is even
1 reference
static bool IsEven(int numberToCheck)
{
    // Check if the number is even
    if (numberToCheck % 2 == 0)
        // Return true if it is even
        return true;
    else
        // Return false if it is not even
        return false;
}
```

Output:




```
using System;
// Program to demostrate user defined methods

namespace Week3Program
{
    0 references
    class MethodDemo
    {
        0 references
        static void Main(string[] args)
        {
            // Variable declaration
            int numToCheck;
            string userInput;
            bool result;
            // Accept input from user
            Console.Write("Enter an integer: ");
            userInput = Console.ReadLine();
            numToCheck = Convert.ToInt32(userInput);

            // Check for Even by passing the userinput to the IsEven() method
            result = IsEven(numToCheck);

            // Check is the result is true ot false
            if (result)
                Console.WriteLine("\nThe Number is Even!");
            else
                Console.WriteLine("\nThe Number is not Even!");

            //Accept a key press from user.
            Console.ReadKey();
        }

        // Create a new method checks whether a number is even
        1 reference
        static bool IsEven(int numberToCheck)...
    }
}
```

Reference: https://www.tutorialspoint.com/csharp/csharp_methods.htm

User Defined Methods in C#

- **Passing parameters to method**:

1. **Value parameters** (default mechanism)

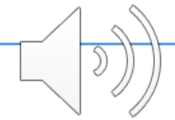
In the example

- numToCheck is the Value passed to IsEven method

```
// Check for Even by passing the userinput to the IsEven() method
result = IsEven(numToCheck);
```

- The value of numToCheck is copied to numberToCheck
- A new storage location is created for numberToCheck, which is a value parameter
- Any Changes made to numberToCheck have not effect on numToCheck

```
// Create a new method checks whether a number is even
1 reference
static bool IsEven(int numberToCheck)
{
    // Check if the number is even
    if (numberToCheck % 2 == 0)
        // Return true if it is even
        return true;
    else
        // Return false if it is not even
        return false;
}
```



Reference: https://www.tutorialspoint.com/csharp/csharp_methods.htm

User Defined Methods in C#

2. Pass Parameter by Reference:

- Unlike value parameters, Reference parameter is a reference to a memory location.
- **ref** keyword is used to declare reference parameters
- A new storage location is not created, they refer to the same memory location as the actual parameter.
- Let us modify the number swap program to pass parameter by reference!

Reference: https://www.tutorialspoint.com/csharp/csharp_methods.htm

User Defined Methods in C#

2. Pass Parameter by Reference: Example

```
// Create a method to swap the values of two variables.
1 reference
public void SwapNumbers(ref int  num1,  ref int num2)
{
    // Declare a temporary variable
    int temp;

    // Swap the values
    temp = num1; // Store the value of num1 in temp variable
    num1 = num2; // Store the value of num2 in num1
    num2 = temp; // Pass the value of temp to num2

    // Display the number After Swaping
    Console.WriteLine("\nInside Swap():");
    Console.WriteLine("After Swaping: number1 = {0}", num1);
    Console.WriteLine("Before Swaping: number2 = {0}", num2);
}
```

Output:

```
Inside Main():
Before Swaping: number1 = 10
Before Swaping: number2 = 20

Inside Swap():
After Swaping: number1 = 20
Before Swaping: number2 = 10

Inside Main():
After Swaping: number1 = 20
After Swaping: number2 = 10
```

```
using System;
// Program to demonstrate the paramenter
// passing by reference

namespace Week3Program
{
    2 references
    class ReferenceParameterDemo
    {
        // Create a method to swap the values of two variables.
        1 reference
        public void SwapNumbers(ref int  num1,  ref int num2)...

        0 references
        static void Main(string[] args)
        {
            //Create an instance of class ReferenceParameterDemo
            ReferenceParameterDemo swap = new ReferenceParameterDemo();

            // Variable declaration
            int number1 = 10, number2 = 20;

            // Display the number before Swaping
            Console.WriteLine("\nInside Main():");
            Console.WriteLine("Before Swaping: number1 = {0}", number1);
            Console.WriteLine("Before Swaping: number2 = {0}", number2);

            // Call the swap method and pass number1 and number2
            swap.SwapNumbers(ref number1, ref number2);

            // Display the number After Swaping
            Console.WriteLine("\nInside Main():");
            Console.WriteLine("After Swaping: number1 = {0}", number1);
            Console.WriteLine("After Swaping: number2 = {0}", number2);

            // Accept a key press from user
            Console.ReadKey();
        }
    }
}
```

Reference: https://www.tutorialspoint.com/csharp/csharp_methods.htm

User Defined Methods in C#

3. Passing parameter by Output

- Similar to ref (reference parameters)
- Transfers data out of the method instead of into the method
- Argument passed using out keyword can be passed without any value assigned it.
- An argument passed using out keyword must be defined in the called method!

Reference: https://www.tutorialspoint.com/csharp/csharp_methods.htm

User Defined Methods in C#

3. Passing parameter by Output

Example

```
// Create a method to accept User Input about student
// using out parameters
1 reference
public void UserInput(out int roll, out int mark, out string name)
{
    // Declare a temporary variable to accept user input
    string tempUserInput;

    // Accept Student name from user
    Console.Write("Enter the Student's name: ");
    name = Console.ReadLine();

    // Accept Students roll from user
    Console.Write("Enter the Student's Roll: ");
    tempUserInput = Console.ReadLine();
    roll = Convert.ToInt32(tempUserInput);

    // Accept Students roll from the user
    Console.Write("Enter the Student's mark: ");
    tempUserInput = Console.ReadLine();
    mark = Convert.ToInt32(tempUserInput);
}
```

Output:

```
Enter the Student's name: Nabin Sharma
Enter the Student's Roll: 1001
Enter the Student's mark: 95

The student information received was:
Student Name: Nabin Sharma,
roll: 1001,
mark: 95
```

```
using System;
// Program demonstrates out parameters

namespace Week3Program
{
    2 references
    class OutParameterDemo
    {
        // Create a method to accept User Input about student
        // using out parameters
        1 reference
        public void UserInput(out int roll, out int mark, out string name)...

        0 references
        static void Main(string[] args)
        {
            // Variable declaration
            int roll, mark;
            string studentName;

            // Create an instance of OutParameterDemo to use the UserInput method
            OutParameterDemo studentData = new OutParameterDemo();
            // Call the user Input method and pass the variables
            studentData.UserInput(out roll, out mark, out studentName);

            // Display the values recieved from the UserInput method
            Console.WriteLine("\nThe student information received was:");
            Console.WriteLine("Student Name: {0}, \nroll: {1}, \nmark: {2}", studentName, roll, mark);

            //Accept a key input from user
            Console.ReadKey();
        }
    }
}
```

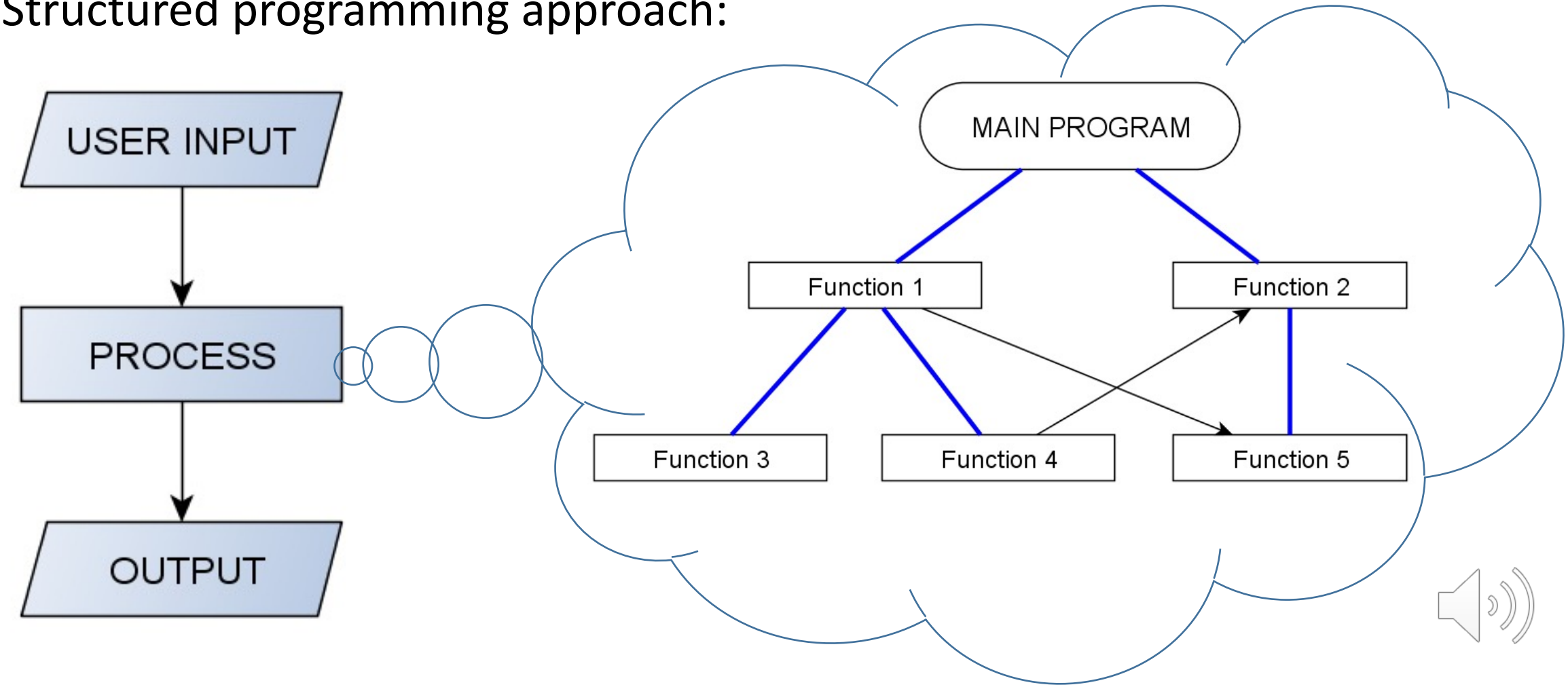
Reference: https://www.tutorialspoint.com/csharp/csharp_methods.htm

Programming Paradigms

- Structured programming approach:
  - Structured Programming is designed which focuses on **process**/ logical structure and then data required for that process. DATA not a priority!
  - Process centric approach, process involved in producing the required output is more important!
  - Programs are divided into small self contained functions
  - Less secure as data hiding is not supported
  - Can be used for moderately complex programming problem
  - Less reusability and more functional dependency

Programming Paradigms

- Structured programming approach:

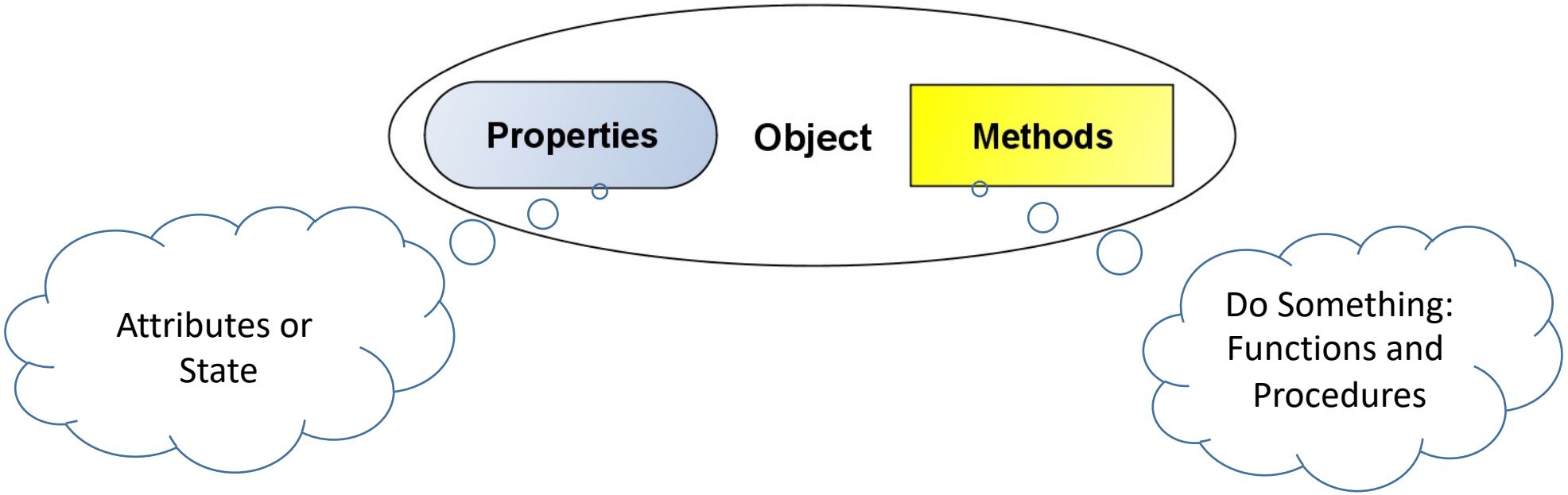


Programming Paradigms

- Object-oriented programming (OOP)
  - Object Oriented Programming focuses on **DATA! and everything about it!**
  - Object Oriented Programming supports **inheritance, encapsulation, abstraction**, **polymorphism**, etc.
  - In Object Oriented Programming, Programs are divided into small entities called **objects**
  - Object Oriented Programming is more secure as having data hiding feature
  - Object Oriented Programming can solve **complex** programming problems
  - Object Oriented Programming provides more reusability, **less function dependency**

Programming Paradigms

- Object-oriented programming (OOP)



Programming Paradigms

- Object-oriented programming (OOP)

Properties:

- Color
- Weight
- Speed
- Seat capacity
- Fuel capacity

Etc.

Methods:

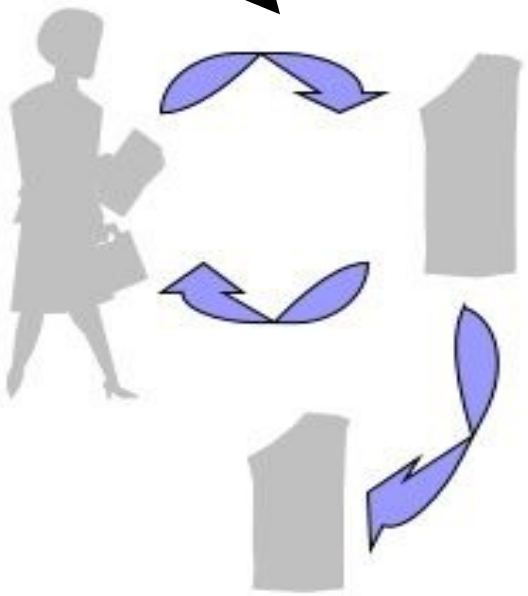
- GoForward
- GoBackward
- turnLeft
- turnRight
- applyBrakes
- activateHorn

Etc.

Image reference: http://images.easyfreeclipart.com/1408/zooming-car-cartoon-logo-1408893.png

Programming Paradigms

- Structured Vs Object-oriented programming example



Withdraw, deposit, transfer

Customer, money, account

Image reference: https://s-media-cache-ak0.pinimg.com/736x/76/c0/df/76c0df8ee6d2a2d38237aa6e33a057c6--coding-procedural-programming.jpg

Object Oriented Programming (OOP)

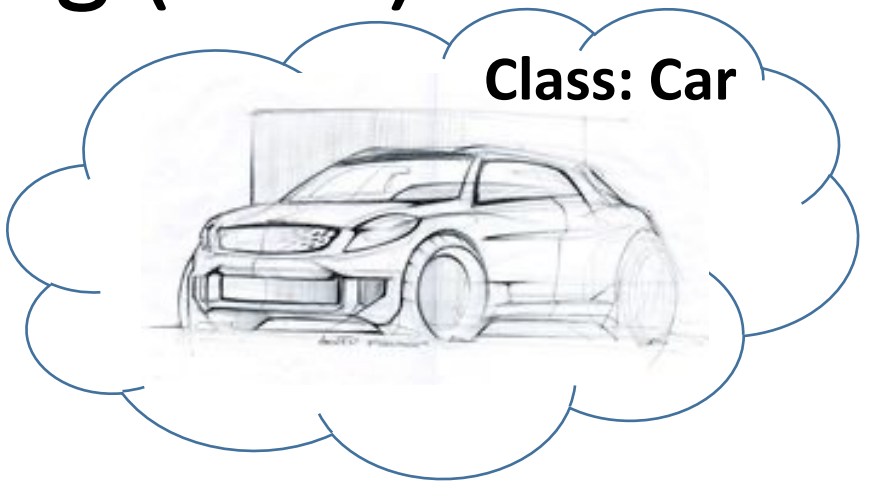
- Object
- Class
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

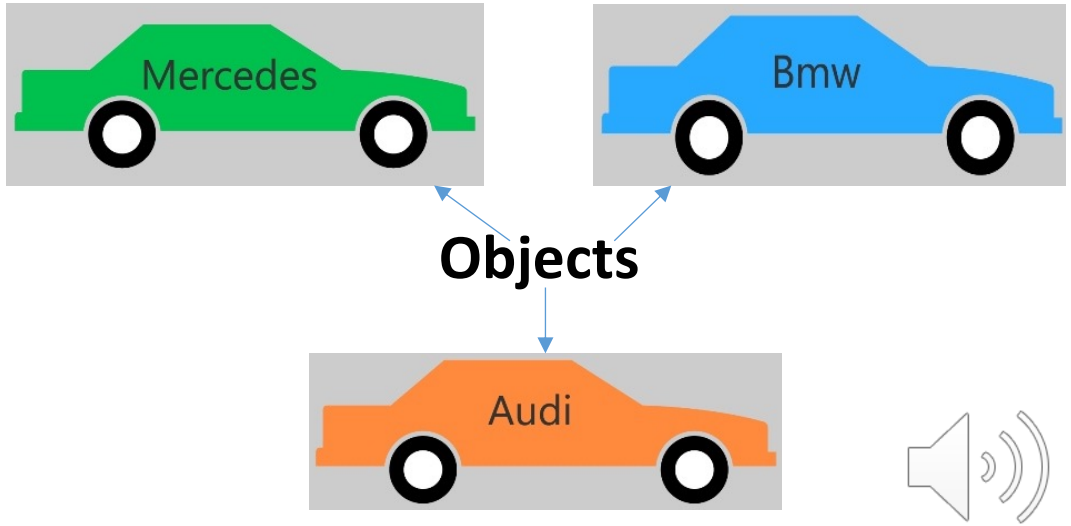
Reference: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/object-oriented-programming

Object Oriented Programming (OOP)

Class and Object

- A class is a “**type**” or a **template**
- Object is an **instance** of class
- Class groups similar objects
  - same (structure of) attributes
  - same services
- Object holds values of its class’s attributes





Object Oriented Programming (OOP)

Class and Object

Class: Student

name
rollNo

setName()
setRollNo()

Objects

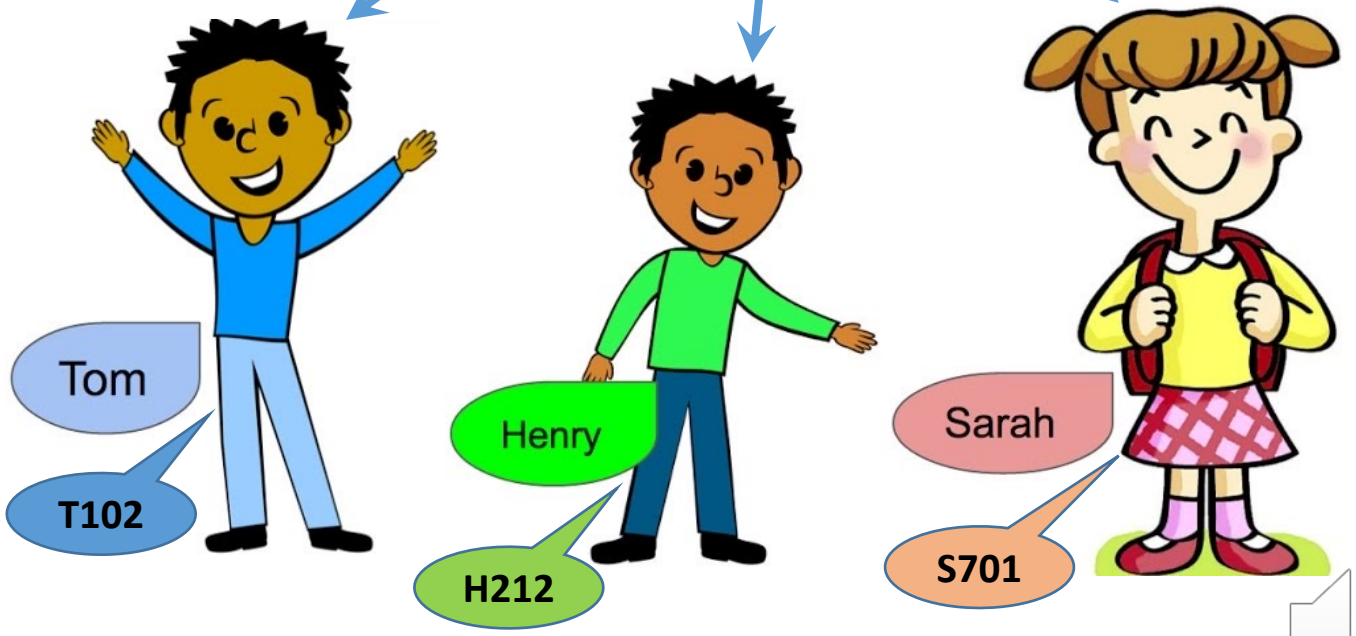


Image reference: https://i.ytimg.com/vi/1wJfIUcVWIA/maxresdefault.jpg

Object Oriented Programming (OOP)

Advantages

- Modularity
  - source code for an object can be written and maintained independently of the source code for other objects
  - easier maintenance and reuse

- Information hiding
  - other objects can ignore implementation details
  - security (object has control over its internal state)

Object Oriented Programming

- **Abstraction**

-Represent the essential feature without implementation details

- Focus on what object does, instead of how it does

- So, what are the essential features of a mobile phone, irrespective of its brand?

- Solves a problem at Design level!

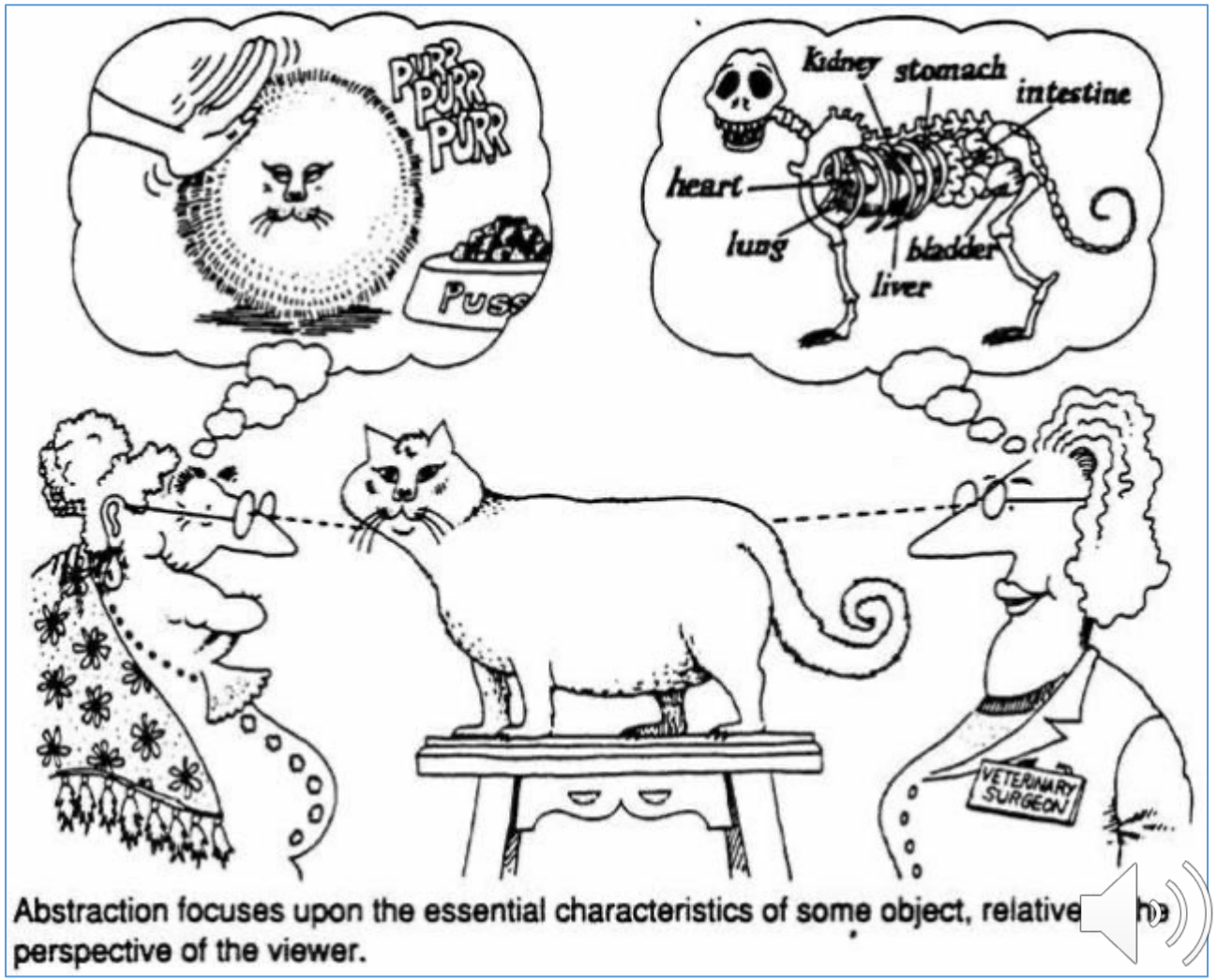


Image Source: https://effectivesoftwaredesign.com/page/3/

Object Oriented Programming

- **Encapsulation**

-Bundling the related functionality together and provide access to only what is needed.

- This is the basis of designing classes

- Everyone outside the cat should see the cat as it is!
- Solves the problem at implementation level

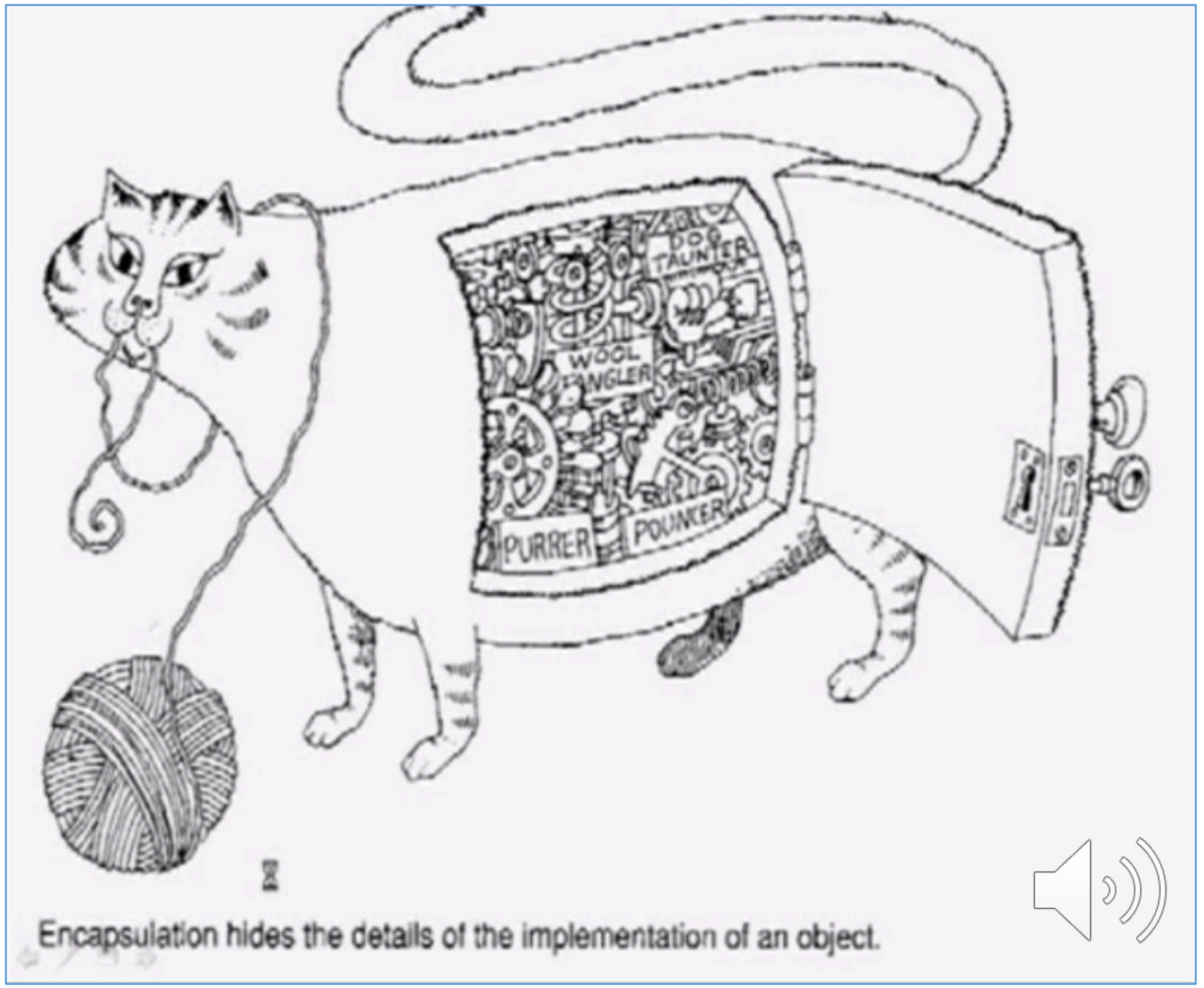


Image Source: https://effectivesoftwaredesign.com/page/3/

Object Oriented Programming

- **Inheritance:**

-A mechanism by which a class acquires some properties and behavior of another class.

- An important concept in Object-Oriented Programming!

- The idea of inheritance implements IS-A relationship

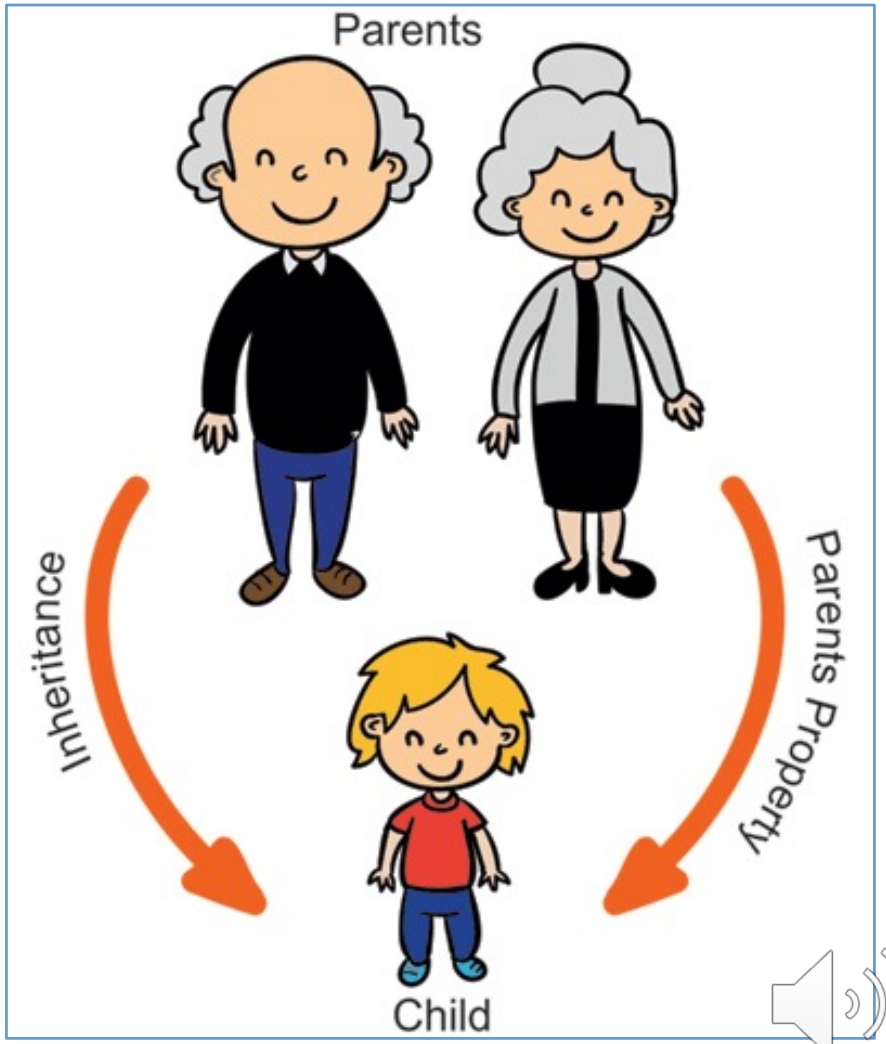


Image Source: https://seleniumautomationtester.wordpress.com/2017/03/09/inheritance-in-java/

Object Oriented Programming

- **Polymorphism:**

 - One thing having many different forms

 - In object-oriented programming: One interface with multiple functions

- Polymorphism can be

    1. Static: Method overloading and Operator Overloading (Compile time)
    2. Dynamic: Abstract class, Method overriding, virtual functions (Run time)

Class in C#

- The keyword class is used to define a class

General syntax:

- <access_modifier> class <class_name> {
  - // Data members (Properties)
    <access_modifier> <data_type> variable_name;
    …
  // Methods
    <access_modifier> <return_type> method_name (parameter_list) {
      // Method body
    }
  …
  }

Reference: https://www.tutorialspoint.com/csharp/csharp_classes.htm

Class in C#

Access modifiers: Keywords used to specify the accessibility of a class/method/type etc.

- public : No restrictions
- private : Access limited to the containing type/class
- protected : Access limited to the containing type/class or derived type/class
- static : Cannot be referenced using an instance, referenced using the type name
- abstract : Used to indicate incomplete or missing implementation
- sealed : Prevents a class from being inherited

Reference: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/access-modifiers

Objects in C#

- In OO-programming, Object are blocks of memory allocated based on the class definition.
- Object in C# are created using the **new** operator
- The new operator create objects and also invokes constructors.

- Example:

```
int number = 0;
```

Is similar to

```
int number = new int();
```

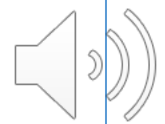
Class and Objects in C#

Example:

```
public void getUserInput()
{
    string userInput;
    Console.Write("Enter the Student's name: ");
    studentName = Console.ReadLine();

    Console.Write("Enter the Student's Roll: ");
    userInput = Console.ReadLine();
    roll = Convert.ToInt32(userInput);

    Console.Write("Enter the Student's Mark: ");
    userInput = Console.ReadLine();
    mark = Convert.ToDouble(userInput);
}
```

```
public void displayInfo()
{
    Console.WriteLine("\nUser Inputs are:");
    Console.WriteLine("Student's name: {0}", studentName);
    Console.WriteLine("Student's roll: {0}", roll);
    Console.WriteLine("Student's Mark: {0}", mark);
}
```

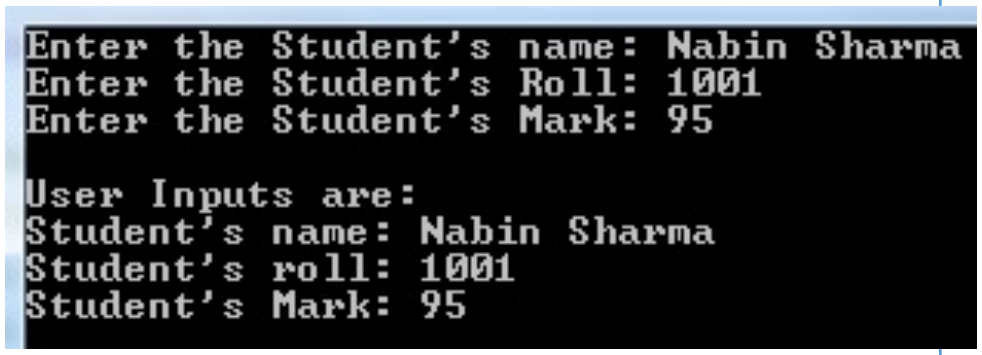
```
using System;
// Program to demonstrate Class

namespace Week3Program
{
    // Declare a Student class with
    // Name, roll and mark are data members
    // A method to accept data from user
    // A method to display the user input
    2 references
    class Student
    {
        // Declare data members
        string studentName;
        int roll;
        double mark;

        // Member Methods
        // Method to accept input from user
        1 reference
        public void getUserInput()...

        // Method to display the values
        1 reference
        public void displayInfo()...
    }
    0 references
    class ClassDemo...
}
```



Class in C#

Example: contd…

Output:




```
namespace Week3Program
{
    // Declare a Student class with
    // Name, roll and mark are data members
    // A method to accept data from user
    // A method to display the user input
    2 references
    class Student...
    0 references
    class ClassDemo
    {
        0 references
        static void Main(string[] args)
        {
            // Create an object of Student Class
            Student student = new Student();

            // Get values from the user
            student.getUserInput();
            // Display the values
            student.displayInfo();

            Console.ReadKey();
        }
    }
}
```

Constructors

- *Constructor* is a special method whose name is same as the class/type name
- *Constructor* doesn't have a return type
- When an object is created its class's constructor is called!
- Every class/type has a *Constructor*
- If an explicit *Constructor* is not provided, C# creates a default constructor (Without any parameters) to instantiate the object.
- *Constructor* set the data members values to there corresponding default values
- Class can have more than one *Constructor*

Reference: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-constructors

Constructors

- Example:

```
class Student
{
    // Declare data members
    string studentName;
    int roll;
    double mark;
```

```
// Default Constructor
1 reference
public Student()
{
    studentName = "Hello World";
    roll = 0;
    mark = 0.0;
}
```

Valid Constructors

```
// Parameterized Constructor
1 reference
public Student(string tempName, int tempRoll, double tempMark)
{
    studentName = tempName;
    roll = tempRoll;
    mark = tempMark;
}
```

Constructors

- Example:

```
class ConstructorDemo
{
    0 references
    static void Main(string[] args)
    {
        // Create an object of Student Class
        Student student = new Student();
        Student student1 = new Student("Nabin Sharma", 1001, 98);

        // Display the values
        student.displayInfo();
        student1.displayInfo();

        Console.ReadKey();
    }
}
```

Output:

```
The values are:
Student's name: Hello World
Student's roll: 0
Student's Mark: 0

The values are:
Student's name: Nabin Sharma
Student's roll: 1001
Student's Mark: 98
```

```
using System;
// Program to demonstrate Constructors

namespace Week3Program
{
    6 references
    class Student
    {
        // Declare data members
        string studentName;
        int roll;
        double mark;

        // Default Constructor
        1 reference
        public Student()
        {
            studentName = "Hello World";
            roll = 0;
            mark = 0.0;
        }
        // Parameterized Constructor
        1 reference
        public Student(string tempName, int tempRoll, double tempMark)
        {
            studentName = tempName;
            roll = tempRoll;
            mark = tempMark;
        }

        // Method to display the values
        2 references
        public void displayInfo()
        {
            Console.WriteLine("\nThe values are:");
            Console.WriteLine("Student's name: {0}", studentName);
            Console.WriteLine("Student's roll: {0}", roll);
            Console.WriteLine("Student's Mark: {0}", mark);
        }
    }
    0 references
    class ConstructorDemo...
}
```