Steps for gui form

1)Open new project name as “gui form 1.cs”

2) form toolbox drop label to gui form1 .cs

3)name label1 as “login here”

4) drop another label and name as user name

5)drop input box for user name and variable name

As text 1

6) drop another label and name as password

7)drop input box for password and variable name

As text box 2

8)drag 2 buttons “cancel and login “respectively

Q1how to connect database with c# program are givern bellow

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MySql.Data.MySqlClient;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

string mysqlconnectString = "datasource=127.0.0.1;port=3306;username=root;password; database=info";

MySqlConnection databasecon = new MySqlConnection(mysqlconnectString);

try

{

databasecon.Open();

MessageBox.Show("Database ok");

}

catch

{

MessageBox.Show("Database Error");

}

}

}

}

Q2) how to made simple log in form (gui)

Ans: Steps for gui form

1)Open new project name as “gui form 1.cs”

2) form toolbox drop label to gui form1 .cs

3)name label1 as “login here”

4) drop another label and name as user name

5)drop input box for user name and variable name

As text 1

6) drop another label and name as password

7)drop input box for password and variable name

As text box 2

8)drag 2 buttons “cancel and login “respectively

Program are given bellow simple form

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace roomgui

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

private void textBox2\_TextChanged(object sender, EventArgs e)

{

}

private void button2\_Click(object sender, EventArgs e)

{

if (textBox1.Text == "Admin" && textBox2.Text == "pass")

{

this.Hide();

Home room = new Home();

room.Show();

}

else

{

MessageBox.Show("Username/Password Incorrect");

}

}

private void button1\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

Q3) how to create login form and connect to data base

Ans

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace roomgui

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

private void textBox2\_TextChanged(object sender, EventArgs e)

{

}

private void button2\_Click(object sender, EventArgs e)

{

string mysqlconnectString = "datasource=127.0.0.1;port=3306;username=root;password; database=info";

MySqlConnection databasecon = new MySqlConnection(mysqlconnectString);

try

{

databasecon.Open();

MessageBox.Show("Database ok");

String user = textBox1.Text;

String pass = textBox1.Text;

Query = "Select \*(count) from employee where username = 'user' and password ='pass'";

if (Query) {

this.Hide();

Home room = new Home();

room.Show();

}

else

{

MessageBox.Show("Username/Password Incorrect");

}

}

catch

{

MessageBox.Show("Database Error");

}

}

private void button1\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

Class example

Example

Create an object called "myObj" and use it to print the value of color:

class Car {

string color = "red";

static void Main(string[] args)

{

Car **myObj** = new Car();

Console.WriteLine(myObj.color);

}

}

## **C# - What is OOP?**

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

* OOP is faster and easier to execute
* OOP provides a clear structure for the programs
* OOP helps to keep the C# code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
* OOP makes it possible to create full reusable applications with less code and shorter developmen

## **C# - What are Classes and Objects?**

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:

## **Class Members**

Fields and methods inside classes are often referred to as "Class Members":

### Example

Create a Car class with three class members: **two fields** and **one method**.

// The class

class MyClass

{

// Class members

string color = "red"; // field

int maxSpeed = 200; // field

public void fullThrottle() // method

{

Console.WriteLine("The car is going as fast as it can!");

}

}

## **Constructors**

A constructor is a **special method** that is used to initialize objects. The advantage of a constructor, is that it is called when an object of a class is created. It can be used to set initial values for fields:

### Example

Create a constructor:

// Create a Car class

class Car

{

public string model; // Create a field

// Create a **class constructor** for the Car class

public Car()

{

model = "Mustang"; // Set the initial value for model

}

static void Main(string[] args)

{

Car Ford = new Car(); // Create an object of the Car Class (this will **call the constructor**)

Console.WriteLine(Ford.model); // Print the value of model

}

}

// Outputs "Mustang"

## **Access Modifiers**

By now, you are quite familiar with the public keyword that appears in many of our examples:

**public** string color;

The public keyword is an **access modifier**, which is used to set the access level/visibility for classes, fields, methods and properties.

C# has the following access modifiers:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| public | The code is accessible for all classes |
| private | The code is only accessible within the same class |
| protected | The code is accessible within the same class, or in a class that is inherited from that class. You will learn more about [inheritance](https://www.w3schools.com/cs/cs_inheritance.asp) in a later chapter |
| internal | The code is only accessible within its own assembly, but not from another assembly. You will learn more about this in a later chapter |

## **Private Modifier**

If you declare a field with a private access modifier, it can only be accessed within the same class:

### Example

class Car

{

private string model;

static void Main(string[] args)

{

Car Ford = new Car("Mustang");

Console.WriteLine(Ford.model);

}

}

The output will be:

Mastang

## **public Modifier**

If you declare a field with a public access modifier, it is accessible for all classes:

### Example

class Car

{

public string model = "Mustang";

}

class Program

{

static void Main(string[] args)

{

Car myObj = new Car();

Console.WriteLine(myObj.model);

}

}

The output will be:

Mustang

[Run example »](https://www.w3schools.com/cs/showjava_classes.asp?filename=demo_mod_public)

### Why Access Modifiers?

To control the visibility of class members (the security level of each individual class and class member).

To achieve "**Encapsulation**" - which is the process of making sure that "sensitive" data is hidden from users. This is done by declaring fields as private. You will learn more about this in the next chapter.

**Note:** By default, all members of a class are private if you don't specify an access modifier:

### Example

class Car

{

string model; // private

string year; // private

}

## **Inheritance (Derived and Base Class)**

In C#, it is possible to inherit fields and methods from one class to another. We group the "inheritance concept" into two categories:

* **Derived Class** (child) - the class that inherits from another class
* **Base Class** (parent) - the class being inherited from

To inherit from a class, use the : symbol.

In the example below, the Car class (child) inherits the fields and methods from the Vehicle class (parent):

### Example

class Vehicle // base class (parent)

{

public string brand = "Ford"; // Vehicle field

public void honk() // Vehicle method

{

Console.WriteLine("Tuut, tuut!");

}

}

class Car : Vehicle // derived class (child)

{

public string modelName = "Mustang"; // Car field

}

class Program

{

static void Main(string[] args)

{

// Create a myCar object

Car myCar = new Car();

// Call the honk() method (From the Vehicle class) on the myCar object

myCar.honk();

// Display the value of the brand field (from the Vehicle class) and the value of the modelName from the Car class

Console.WriteLine(myCar.brand + " " + myCar.modelName);

}

}

[Run example »](https://www.w3schools.com/cs/showjava_classes3.asp?filename=demo_inheritance)

#### **Why And When To Use "Inheritance"?**

- It is useful for code reusability: reuse fields and methods of an existing class when you create a new class.

## **Polymorphism and Overriding Methods**

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like we specified in the previous chapter; [**Inheritance**](https://www.w3schools.com/cs/cs_inheritance.asp) lets us inherit fields and methods from another class. **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example, think of a base class called Animal that has a method called animalSound(). Derived classes of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

### Example

class Animal // Base class (parent)

{

public void animalSound()

{

Console.WriteLine("The animal makes a sound");

}

}

class Pig : Animal // Derived class (child)

{

public void animalSound()

{

Console.WriteLine("The pig says: wee wee");

}

}

class Dog : Animal // Derived class (child)

{

public void animalSound()

{

Console.WriteLine("The dog says: bow wow");

}

}

## **Abstract Classes and Methods**

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.  
Abstraction can be achieved with either **abstract classes** or [**interfaces**](https://www.w3schools.com/cs/cs_interface.asp) (which you will learn more about in the next chapter).

The abstract keyword is used for classes and methods:

* **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

* **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).

An abstract class can have both abstract and regular methods:

abstract class Animal

{

public abstract void animalSound();

public void sleep()

{

Console.WriteLine("Zzz");

}

}

## **C# Enums**

An enum is a special "class" that represents a group of **constants** (unchangeable/read-only variables).

To create an enum, use the enum keyword (instead of class or interface), and separate the enum items with a comma:

### Example

enum Level

{

Low,

Medium,

High

}

You can access enum items with the **dot** syntax:

Level myVar = Level.Medium;

Console.WriteLine(myVar);

## **Enum Values**

By default, the first item of an enum has the value 0. The second has the value 1, and so on.

To get the integer value from an item, you must [explicitly convert](https://www.w3schools.com/cs/cs_type_casting.asp) the item to an int:

### Example

enum Months

{

January, // 0

February, // 1

March, // 2

April, // 3

May, // 4

June, // 5

July // 6

}

static void Main(string[] args)

{

int myNum = (int) Months.April;

Console.WriteLine(myNum);

}

The output will be:

## **# Exceptions**

When executing C# code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, C# will normally stop and generate an error message. The technical term for this is: C# will throw an **exception** (throw an error).

## **C# try and catch**

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs:

### Syntax

try

{

// *Block of code to try*

}

catch (Exception e)

{

// *Block of code to handle errors*

}

## **inally**

The finally statement lets you execute code, after try...catch, regardless of the result:

### Example

try

{

int[] myNumbers = {1, 2, 3};

Console.WriteLine(myNumbers[10]);

}

catch (Exception e)

{

Console.WriteLine("Something went wrong.");

}

finally

{

Console.WriteLine("The 'try catch' is finished.");

}

## **The throw keyword**

The throw statement allows you to create a custom error.

The throw statement is used together with an **exception class**. There are many exception classes available in C#: ArithmeticException, FileNotFoundException, IndexOutOfRangeException, TimeOutException, etc:

### Example

static void checkAge(int age)

{

if (age < 18)

{

throw new ArithmeticException("Access denied - You must be at least 18 years old.");

}

else

{

Console.WriteLine("Access granted - You are old enough!");

}

}

static void Main(string[] args)

{

checkAge(15);

}

­