# Objective

* To apply basic object oriented principles to solve problems by writing code
* To exercise logic by supplying appropriate C# code to solve a problem

# Principles:

* Anatomy of a C# class
* Creating and instantiating classes into objects
* Creating private attributes and public methods
* Control flow statements
* Basic inheritance

## Exercise 1

1. Create a new dotnet console project called Exercise1
2. Create a class called SimCard with the following member attributes:
   1. phoneNumber (string)
   2. network (string. Example of network can be “Globe” or “Smart”)
3. Provide a constructor for SimCard that will always accept a phoneNumber and network when being instantiated.
4. Supply the necessary methods that you think a SimCard class might need.
5. Create a MobilePhone class with the following attributes:
   1. brand (string)
   2. color (string)
   3. simCard (SimCard)
   4. model (string)
6. Create a constructor that instantiates a MobilePhone and passes all 4 attributes.
7. Create a method in MobilePhone called GetPhoneNumber() that returns the phone number of the phone’s sim card.
8. Create a method in MobilePhone called SwapSimCard(SimCard simCard) that replaces the sim card of the phone.
9. Create a method in MobilePhone called Call(string phoneNumber) that displays the following text: “Calling [phoneNumber] from number [GetPhoneNumber()’s value].
10. Overload the method Call by creating another implementation Call(MobilePhone mobilePhone) which produces the same output but the phone number is taken from the passed mobilePhone’s sim card.
11. Create a class called App that has the following attributes:
    1. name (string)
    2. size (float)
12. Create a class called SmartPhone that inherits MobilePhone.
13. Create the following attributes in SmartPhone:
    1. browser (string)
    2. applications (List<App>)
14. When a SmartPhone is created, it should invoke a constructor that passes a browser value together with other attributes needed by MobilePhone
15. Create a method for SmartPhone called CallFromBrowser(string phoneNumber) that first displays the text “Calling from [browser]” where [browser] is the browser of the phone and then invoking Call(string phoneNumber) of its parent.
16. Overload the method CallFromBrowser and supply another implementation called Call(MobilePhone mobilePhone) that does the same thing in number 15 but instead invokes its parent’s Call(MobilePhone mobilePhone) method.
17. Create a method called AddApplication(App app) for SmartPhone that adds the application to the SmartPhone’s applications list.
18. Create a method called GetTotalSize() that returns the total size of applications in a smart phone.
19. Supply other utility methods for SmartPhone (this is up to you)
20. In Program.cs, write code that will demonstrate at least the following:
    1. 2 instances of MobilePhone
    2. 2 instances of SmartPhone
    3. MobilePhone talking to another MobilePhone
    4. SmartPhone talking to another SmartPhone
    5. Adding applications to a SmartPhone
    6. Displaying the total size of the applications of a smart phone

## Exercise 2

1. Create a class called TodoItem with the following attributes:
   1. id (integer)
   2. content (string)
   3. status (string. Possible values: “pending”, “active”, “done”)
2. Create a constructor for TodoItem with the signature TodoItem(int id, string content). The constructor should set the attributes of id and content respectively with a default status of “pending”.
3. Create a method in TodoItem called update() with a bool return type. The logic for update will be as follows:
   1. If the status of the item’s status is pending, then update it to active. The method then returns true.
   2. If the item’s status is active, then update it to done. The method then returns true.
   3. If the item’s status is done, then do not do anything and return false.
4. Create a class called TodoList that has the following attributes:
   1. id (integer)
   2. name (string)
   3. todoItems (List<TodoItem>)
5. Create a constructor for TodoList that accepts the id and name, and sets up its attributes accordingly. The todoItems should initially be an empty list.
6. Create a method in TodoList called AddTodoItem(TodoItem item) that adds a new item to its todoItems.
7. Create a method in TodoList called RemoveTodoItem(int id) that removes an item based on the id passed to it.
8. In Program.cs, create an instance of List<TodoList> using a variable called myLists.
9. In your Program.cs create an interface that shows the following options:
   1. 1 - Display All Lists
   2. 2 - Show Items
   3. 3 - Create New List
   4. 4 - Select List
   5. 5 - Quit
10. Ask the user to enter a number corresponding to the choice. It should have the following logic:
    1. If 1 is selected, it displays all the current TodoLists in myLists. The interface should display the id and name of a list together with the number of items.
    2. If 2 is selected, as the user to enter an id of a list. If the list is not found, display the original menu again. If found, then display all the items in the list. If there are no items in the list, the program should display “No items found for list [name of list]”.
    3. If 3 is selected, ask the user to enter a name and create a new instance of TodoList to be added to myLists. The id of the list should be 1 number higher than the previous list in myLists. If there are no current lists in myList, the starting id is 1.
    4. If 4 is selected, the program should first ask for the id of the list that is to be selected. If a list is not found, display the origins menu again. If a list is found, display another menu with the following:
       1. 1 - Display all Items
       2. 2 - Create New Item
       3. 3 - Delete Item
       4. 4 - Update Item
       5. 5 - Go back
    5. If 1 is selected in this secondary menu, then display all items in the list. It should show the id, content and status of the item.
    6. If 2 is selected, ask the user to input content for the new item and create an instance of TodoItem for the list. Take note that the id value should be one number higher than the previous item in the items of the list. If no items are in the list then default to 1. After creating, the program should go back to the secondary menu (item d in logic).
    7. If 3 is selected, ask the user to input the id of the item to delete. If an item is found, it should delete that item from the list. If not, display “Invalid id” and go back to the secondary menu (item d in logic).
    8. If 4 is selected, ask the user to input the id of the item to delete. If an item is found, invoke the update() method of that item changing its status. If an item is not found, display “Invalid id” and go back to the secondary menu (item d in logic).
    9. If 5 is selected, go back to the original menu.
    10. If 5 is selected in the original menu, then terminate the program.