C# Lab Works

LAB 1 : BASIC OOL CONCEPTS 1

EXERCISE 1 : STATIC MEMBERS 1
EXERCISE 2 : PROPERTIES 1
EXERCISE 3 : SIMPLE INHERITANCE 2
EXERCISE 4 : THREADS 2

LAB 2 : DELEGATES, EVENTS, THREAD-SAFE CONTROL ACCESS, RESOURCE MANAGEMENT 3

EXERCISE 1 : DELEGATES/EVENTS 3
EXERCISE 2 : THREADS CONTINUED, USING FORMS 3
EXERCISE 3 : RESOURCE MANAGEMENT 4

LAB 3 : INHERITANCE, POLYMORPHISM, CONTAINERS, SERIALIZATION 5

EXERCISE 1 : INTERFACES 5
EXERCISE 2 : DYNAMIC LINKING/POLYMORPHISM, SERIALIZATION 5
EXERCISE 3 : EXERCISE 1 REVISITED 5

LAB 4 : SYNCHRONIZATION 7

EXERCISE 1 : DISCOVERING THE RACE PROBLEM 7
EXERCISE 2 : MULTITHREADED COMPUTATION 7

Lab 1 : basic OOL concepts

Exercise 1 : static members

Write a class named X containing a static attribute allowing to count the number of objects of this class created in a program. Furthermore, override the `ToString()` method for this class X so that the following program :

```
static void Main(string[] args)
{
      X obj1, obj2;

      obj1 = new X();
      obj2 = new X();

      Console.WriteLine(obj1);
      Console.WriteLine(obj2);
}
```

Produces the following output :

```
Object from class X (number 1)
Object from class X (number 2)
```

Exercise 2 : Properties

a) Write a class storing a `private long` attribute. Write a property to access and change this attribute so that this attribute is always strictly positive.

b) Write a class storing two `private long` attributes named `_min` and `_max`. Write a default constructor (with no arguments) to initialize `_min` to 0 and `_max` to 100. Write two properties to access and change `_min` and `_max` so that `_min` is always less or equal than `max`.

Exercise 3 : simple inheritance

Write a class named `instrument`, declaring the abstract method play(). From this class, inherit the class `brass`, declaring the virtual method blow(). From the brass class, inherit the class `trumpet`, implementing both play() and blow(). (for your information, you must blow into a trumpet in order to play some sound, more rarely music…).

Your program must show the order in which all the constructors and methods are called.

Pay attention to accessibility/visibility/scope in this exercise : in the Main() method, the only method a trumpet object can call is : play().

The Main method should look like this :

```
static void Main(string[] args)
{
      new trumpet().play();

      // pause if needed
}
```

Exercise 4 : Threads

Create a "console application" C# project. Write a program that creates 5 threads, each of these threads displays a letter 'A' for the first one, 'B' for the second, and so on. Modify the program so that each thread displays its letter in an infinite loop.

You can add a pause inside a thread by using the `System.Threading.Thread.Sleep(int milliseconds);` method. The Sleep() method blocks the thread but keeps it in execution state.

Lab 2 : Delegates, events, Thread-safe control access, resource management

Exercise 1 : Delegates/Events

Here is a PrimeEventArgs class inheriting from EventArgs. This class stores a long value, and a read-only Property to access this value :

```
public class PrimeEventArgs : System.EventArgs
{
	private long _prime;

	public long Prime
	{
		get {return _prime;}
	}

	public PrimeEventArgs(long p):base()
	{
		_prime = p;
	}
}
```

Write a PrimeSender class raising a PrimeEventArgs event.

Write a PrimeReceiver class with a `public void action(object sender, PrimeEventArgs pe);` method displaying the long value stored in the event received. This object must suscribe to the event raised by the PrimeSender class.

In PrimeSender, write a CalculatePrimes(long limit) method that calculates prime numbers from 1 to limit and raises a PrimeEventArgs event whenever it finds one.

Exercise 2 : Threads continued, using Forms

Step 2) Create a "windows application" C# project. Write a program that creates 5 threads, each of these threads displays a letter ('A' for the first one, 'B' for the second, and so on) once. Modify the program so that each threads displays its letter in an infinite loop.

The main form must contain a button to start the threads and a textBox (set the multiline property to true). The threads must display the letters they write in the textbox. What happens when executing the program ? Try to find how to solve this thread unsafe access (check VS help related to the execution error : InvalidOperationException).

Exercise 3 : Resource management

Create a class systemResource used to modelise a computer system resource. This class must have an attributed name indicating the type of the resource (printer, ram, disk, and so on…).

Create a class askResource used to modelise a computer process. An askResource object can use a computer system resource for a certain amount of time. During this time, this resource is not available to other askResource objects.

An askResource object is simply a list of necessary resources and their timing. For example, an askResource object can ask for a disk systemResource during 5 s, then for ram during 10s, then for printer for 30s.

The askResource class must implement the two following methods : acquireResource(…); and freeResource(…);

Write a program creating a set of systemResources objects, and several askResource objects. The program ouput must show the "execution" of askResources objects, and indicate when the resources they ask for is not available.

Lab 3 : Inheritance, polymorphism, containers, Serialization

Exercise 1 : Interfaces

Create an IAnimal interface declaring the `move()` and `eat()` methods, and storing a `_name` attribute accessible through a `name` property.

Create the IMammal and IReptile interfaces deriving from IAnimal, try to declare in those interfaces specific methods. What should you do if a method appears to be necessary to both IMammal and IReptile classes ?

In a third step, create several classes, where each class represents an animal (lion, cow, snake, lizard, platypus).

Exercise 2 : Dynamic Linking/Polymorphism, Serialization

Add a zoo class that acts as a container for animals. This class must provide the following functionalities :

make all animals have a walk;

make all animals eat;

Serialize all the animals stored in a zoo object

Deserialize all the animals in a zoo object

Test your classes by adding animals to a zoo object and by calling the functions.

Exercise 3 : Exercise 1 revisited

Change your previous implementation of Exercise 1 by replacing the interfaces with normal (non abstract) classes.

Create an Animal class declaring the `move()` and `eat()` methods, and storing a `_name` attribute accessible through a `name` property. Display different strings when these functions are called (“Animal is moving” or something similar).

Create the Mammal and Reptile classes deriving from Animal, override the methods with different strings displayed.

In a third step, do not change the implementations of the subclasses lion, cow, snake, tc.

What happens when you run the code from Exercise 2 in this case ?

Is it different from the behaviour of your previous implementations using interfaces ? In the case there are differences, what needs to be done to make their behaviour similar ?

Lab 4 : Synchronization

Exercise 1 : Discovering the Race Problem

Let "i" be a shared integer initiazed to 5. Assume two threads (T1 and T2) running and respectively incrementing and decrementing "i".

1. run both threads and display the final value; is the result the one you were expecting ?
2. Implement the following algorithm and explain why it could result in 4 or 6

   ```
   Reg = i
   sleep for some time
   Reg++ (or Reg--)
   i=Reg
   ```

3. Solve this problem by enforcing mututal exclusion using semaphores

Exercise 2 : Multithreaded Computation

Use semaphores to implement the following parallelized calculation (a+b)*(c-d)*(e+f) as described below :

T1 runs (a+b) and stores the result in a shared table (1st available spot)
T2 runs (c+d) and stores the result in a shared table (1st available spot)
T3 runs (e+f) and stores the result in a shared table (1st available spot)

T4 awaits for any two tasks to end and does the corresponding calculation (use join or flags)
T4 awaits for the remaining task to end and does the final calculation then displays the result