PCS4 Exam – June 2015

Date: 17 June 2015
Time: 13. 15 – 16.15 h (3 hours)

ADMITTED RESOURCES:

- You are allowed to use everything on paper (books, notes, etc.) and on your laptop, but only what you bring in: you are not allowed to borrow something from someone else.
- During the exam it is not allowed to use the network. You should make the exam yourself: so no communication with MSDN or google for help and no communication with other students, like facebook, e-mail, skype, gsm or whatever.

GRADING:

EXERCISE:	1	2	3	4	5	**6**
POINTS:	15	15	20	20	20	**10**

Assignment 6 can be skipped in case you have been allocated as an 'active' student for PCS4 this block. You will get the corresponding 10 points for this assignment as a bonus.

PRELIMINARY REMARKS:

- Whenever this exam paper suggests to use a certain name for a method, variable or anything else, you are required to indeed use that name.
- It is not allowed to add or remove controls on the form.
- You are allowed to use all predefined methods for lists and arrays unless otherwise is specified in an assignment.

THE ASSIGNMENTS

Given is an application to be used by athletes (runners) to store and compare their performances. These athletes regularly run a track of fixed length and their achieved times (in whole seconds) are stored in a list. For every runner there is a separate list. Whenever a new race has taken place every participant adds his new time at the end of his list. (So the order of the lists will never change, only new values are added at the end).

There are several classes defined in the project, but at this moment there is only one class that needs some explanation. That is the class Runner that matches the following class diagram:

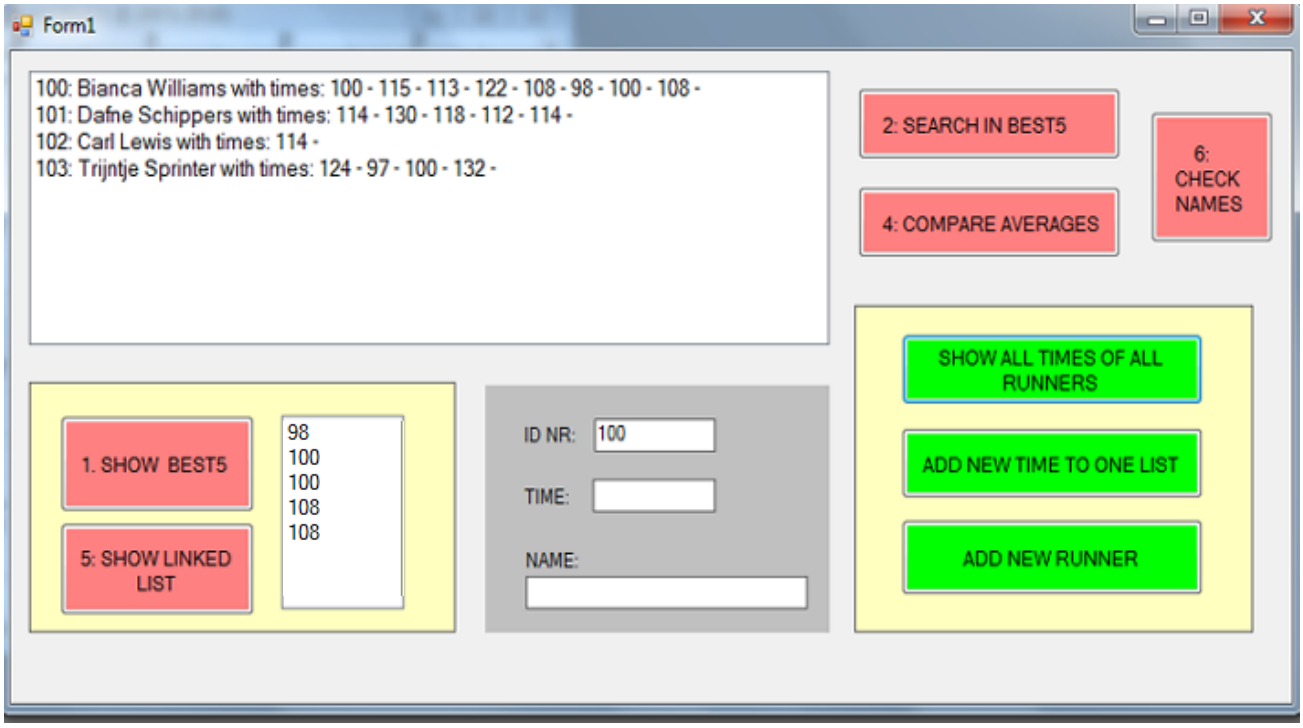
Runner
+<<property>> IdNr : int +<<property>> Name : string +<<property>> MyTimes : List<int> +<<property>> Best5 : int[] – nextIdNr : int
+ Runner(string name) + ToString ( ) : string + AddNewTime (int i) : void

Each `Runner`-object has a unique `IdNr`. Furthermore, each runner has a name, and a list of his achieved times (in the order they are achieved).

The `Runner` class also contains an array (`Best5`), to keep track of the 5 best times achieved by the runner. At this moment the array is not yet used, so all the elements in this array are initialized to 1000. (You may assume that real times will always be less than 1000).

Finally, the method `AddNewTime` enables to append (= add at the end) a new time to the list.

Once you open the project you can see the form with a lot of controls on it (see the picture below). The green buttons are already implemented, the red buttons will be implemented in the course of this exam.



In the `Form1`-class a list of `Runner` objects has already been defined as an instance variable (named `runners`) and in the constructor of the `Form1`-class several runners are added to that list together with several “fake” times.

EXERCISE 1 *(12 + 3 points)*

In this assignment we ask you to implement the button-Click "SHOW BEST5". This button must show the 5 values contained in the array Best5 for a certain runner. But at this moment these arrays only contain the value 1000.

a. Therefore you first need to write the method `FillBest5` in the `Runner` class. This void method (without any parameters) must fill the array `Best5` (which is an array of length 5) with the 5 lowest values of the list `MyTimes`. This must be done in increasing order (so the lowest at index 0) and duplicates are allowed in the array. In this method it is NOT allowed to change the order of the original list `MyTimes`! And it is not allowed to use predefined methods like Sort() .
Note: If the list contains less than 5 values, the remaining elements of the array remain 1000.

b. Now add functionality to the button "SHOW BEST5". This button must first read the idnr contained in the textbox `tbIdNr`. Then the button must display for this runner all the 5 values of his array `Best5` in the Listbox `lbBest5`. Make use of the method written in a. to assign the correct values to the array before it is displayed.

EXERCISE 2 *(10 + 5 points)*

The button "SEARCH IN BEST5", must find all the runners who have a given time in their `Best5` array. So we ask you to:

a. Write in the `Runner` class a method `SearchInBest5`, with one integer (a time) as a parameter . This method must search in the `Best5` array whether it contains that time . If so the method returns true otherwise it returns false.
Although array `Best5` is quite short, you must apply the binary search algorithm for this searching. And it is not allowed to make calls to other methods (neither predefined nor own-written methods). So put all the code immediately in method `SearchInBest5` itself.

b. Now add functionality to the button "SEARCH IN BEST5" . Before clicking this button the user must enter a time (in `tbTime`). This values is used to search in the array `Best5` of each runner. All the names of the runners with this value in their `Best5` array are displayed in the listbox `lbAll`.
Note: To be sure that the array `Best5` contains the correct values, again make use of the method `Fillbest5` (written in assignment 1) for each runner before you start searching in his `Best5` array.

EXERCISE 3 *(7 + 7 + 6 points)*

a. We ask you to add an event to the `Runner` class (with the name `NewRecord`). This event must be raised whenever the runner achieves a new personal record. So that means whenever list `MyTimes` is extended by a new value that is less than all the values in the list until now. While raising this event, a reference to the runner object and the new achieved record should be passed as parameters. Note: In case it is the first value added to the list we also consider it as a personal record so in that situation the event must also be raised.

b. In the `Form1` class you must add a handler (named `Handler1`) for this event. This handler must carry out two things:
   (1) display a `MessageBox` containing the name of the runner who achieved the new record together with that new record. (For example a message like "time 24 is the new record of runner Carl Lewis".)
   (2) display the <u>new</u> content of the `Best5` array for that runner in the listbox `lbBest5`. Of course this new content must first be calculated.

c. Make sure that as soon as the program starts the handler is attached to each runner created at the start of the program. But this handler must also be attached to every runner created later on (see button "ADD NEW RUNNER") .

EXERCISE 4 *(15 + 5 points)*

In this assignment we are going to compare the averages of the `MyTimes` of all the runners. We want to find the runner with the lowest average and the runner with the second lowest average.

a. First write a void method `CompareAverages` in the `Form1` class to compare all the MyTimes-averages. The method must return (by parameters) these 2 averages (the best and the second best) together with the names of the runners that belong to them. (So in total that are 4 values to return.)
Note: Runners with an empty list of times must be ignored here. You may assume that there are at least two runners who don't have an empty list.

b. Now use this method to implement the "COMPARE AVERAGES " -button. After clicking this button a `MessageBox` must be displayed with all the information returned by this method . So the `MessageBox` must contain a string like "the best runner is Bianca Williams with an average of 108 and the second best runner is Trijntje Sprinter with an average of 113,25".

EXERCISE 5 *(3 + 10 + 7 points)*

We want to convert all the `Best5` arrays into linked lists . Therefore we already defined 2 new classes: `TimeNode` and `LinkedTimeList` (both in the file `LinkedList.cs`). Together they (partly) implement a linked list of time-values.

a. Because every runner needs his own linked list, extend the `runner` class in such a way that each runner gets his own `LinkedTimeList` object. This object must be empty at the start and must be made accessible for other classes.

b. The class `LinkedTimeList` doesn't have a method yet to add values to the linked list. We don't want a method to add one single values but we need a method to add all the values from a `Best5` array. We ask you to write this (void) method (called `ConvertBest5ToLinkedList`) with an integer array as parameter. You may assume that this parameter is always a `Best5` array, so with a length of 5. The method must copy the values from that array to the linked list except the values 1000 which were used to completely fill `Best5` if there were not enough real time-values . The values must be arranged in exactly the same order as in the array.

c. Finally add functionality to the button "SHOW LINKED LIST". Whenever this button is clicked, the idnr entered in `tbIdNr` must be used to select a runner. For this runner implement the following actions:
- update his `Best5` array
- convert it to a linked list
- display all the values now contained in that linked list in the listBox `lbBest5`.

EXERCISE 6 *(10 points)*

In this last exercise we ask you to check whether there are multiple runners with the same name. This must be done with recursion.

a. First write a recursive method (in the `Form1` class) to find such a name in the runners list that is used twice or more. If there is such a name the method returns that name (a string). If all the names are different, it returns null. Note: you don't have to find all the names that are used twice or more. As soon as you find one such name the method can return that name and stop.

b. Use this method to implement the button “CHECK NAMES” and display the name found (or a message that such a name is not available) in a `MessageBox`.