Software developer (.NET / C#)

Technical interview questions

Lunes R&D Ltd.

August 13, 2019

Guidelines

- There is no time limit for these questions, but we would like you to hand in your answers within one week.
- You can use Google, Stack Overflow, etc. as you please. The focus is on ability, not raw fact knowledge.
- Don't just make the programs work. The more important criteria are
  - Stability
  - Readability
  - Modularity
  - Extensiblity
  - Performance
- You may use the latest C# / .NET version and features.
- Write your identifier names and comments in english.

If you have any questions about the tasks, feel free to contact stefan.lutz@lunes.de.

1 Basics

This basic tasks is about knowledge of elementary programming language features and familiarity with the IDE.

1.1 Create a new console application.

1.2 Print all numbers from 1 to 100 to the console, in ascending order.

1.3 Print all numbers from 1 to 100 to the console in descending order.

1.4 Calculate the sum of numbers from 1 to n

Write a method *Sum* as follows

```
public static int Sum(int n)
{
    // TODO ...
}
```

that computes $\sum_{i=1}^{n} i$, the sum of all numbers from 1 to n.
Note: Overflow does not need to be taken into account.

1.5 Program two alternative versions of sum

There are several ways to implement the method described in the previous question. Write two differing implementations

```
public static int Sum_A(int n)
{
// TODO ...
}

public static int Sum_B(int n)
{
// TODO ...
}
```

with the same functionality, but using different language features. You should choose a total of 3 from the following:

- Implementation using a for-loop.
- Implementation using a while-loop.
- Implementation using LINQ.
- Implementation that runs in constant time $O(1)$.

2 Simple image processing

Complete a simple unfinished UI application which allows the application of image filters and displays the results.

2.1 Open the project 02_BitmapPlayground

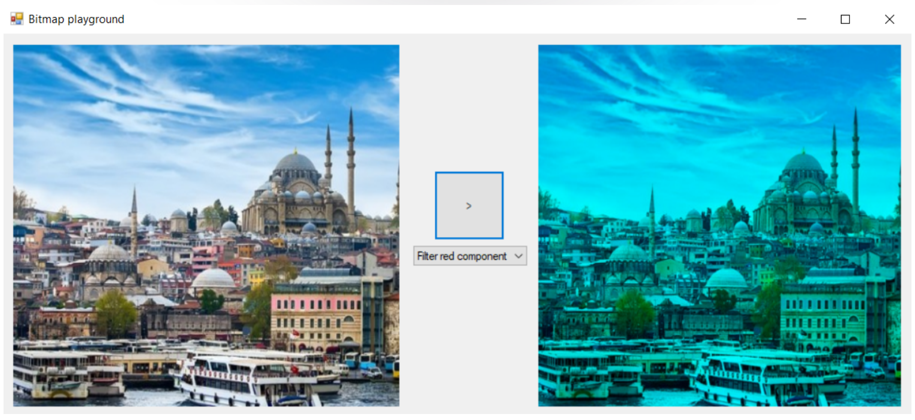


Figure 1: The application with the already implemented red filter.

Familiarize yourself with the project. There is already one filter function available that removes the red component from the image. Try it out. Identify the parts marked with “TODO” in the source code.

2.2 Implement a greyscale filter

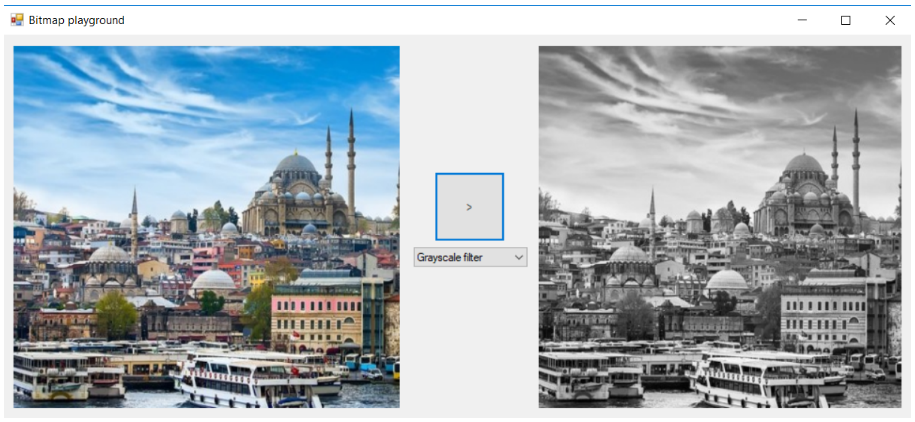


Figure 2: The desired grayscale filter effect.

Implement a greyscale filter. There are several possibles formulas for computing greyscale images, you are free to choose any.

2.3 Implement a moving average filter

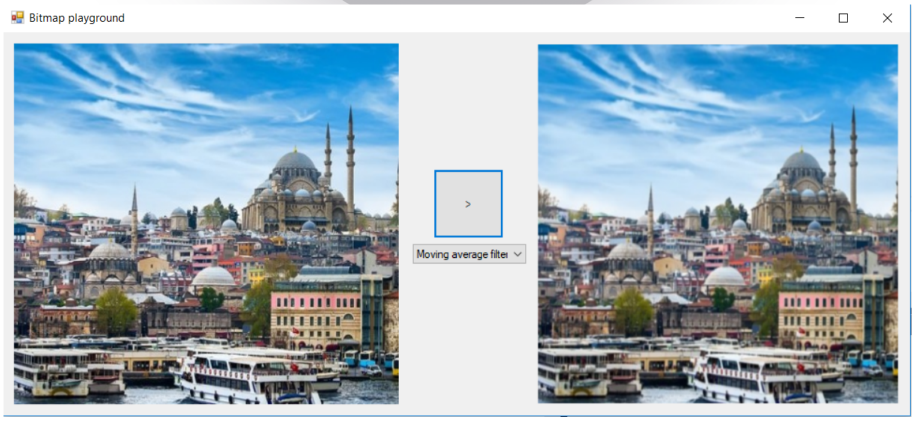


Figure 3: The desired moving average filter effect.

Implement a moving average filter, as follows: The value of each output pixel at the position $output(x, y)$ is the average value of the input pixel $input(x, y)$ and the 4 pixels left, right, above and below it. If the input pixel lies at the border, its value may remain the same.

2.4 Parallelize the filter

Optimize the filters by using thread pooling.

2.5 Extract the filters to an external project

Extract `IFilter` and all of its implementations to an a separate dll project and reference it from the main project.

2.6 Automatically find all filters in the assembly using reflection

Implement `PopulateListBox()` such that all implementations of `IFilter` in the assembly containing `IFilter` are found and added to the dropdown. The text in the Listbox should correspond to their "Name" property.

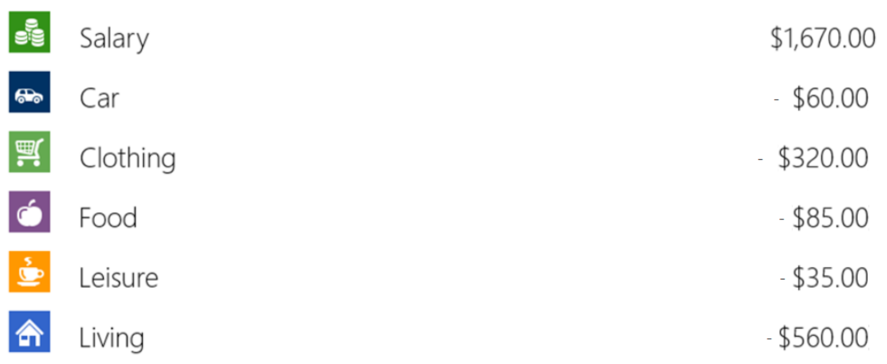
3 WPF / MVVM

Create a small WPF application according to a specification. Use proper separation of concerns and data binding throughout the process. A nice design of the application is a plus, but not essential.

If you are more familiar with web development, you may use ASP.NET or Blazor as an alternative to WPF.

The application should manage the list of monthly transactions of the user. These can be negative (spendings, such as rent) or positive (earnings, such as salary).

3.1 Create a new WPF application



Salary	$1,670.00
Car	- $60.00
Clothing	- $320.00
Food	- $85.00
Leisure	- $35.00
Living	- $560.00

Figure 4: An example for the list view. Icons, colors, design etc. is not essential and up to you.

Setup the application and possibily necessary references / packages you might need. You can use existing MVVM / MVC frameworks or lay out the classes by yourself.

3.2 Add a list of the items

The content of each column should be editable. Use validation to check if the entered text for the amount is actually a number.

3.3 Insert add and delete buttons

Insert two buttons for adding and deleting earnings or spendings.

3.4 Save and load the data to a file or database

Add two buttons (for example in a menu) for saving and loading the data. The choice of the data storage is up to you. You can use SQL with ADO.NET, EF, file formats XML or JSON or a similar datasource.

3.5 Localization

Localize the application using resources in english and turkish.

4 Algorithms

The task is to write a managing system for the conversion of arbitrary files, possibly using several intermediate steps. You are provided with an interface for a file converter

```
public interface IFileConverter
{
string InputFileFormat { get; }
string OutputFileFormat { get; }
object Convert( object input );
}
```

The actual implementation of the converter is irrelevant here. Please do not implement any specific converters! For example, just imagine there being a converter `Png2jpgConverter` which takes an image of type .png and converts it to one of type .jpg and a converter `Jpg2BmpConverter` which converts jpg → bmp.

4.1 Implement a conversion manager

Implement a static class `FileConversionManager` (or similar) which holds a list of known `IFileConverter` objects. It should provide a method for registering file converters.

4.2 Implement the conversion method

The conversion manager should have one main method.

```
public static object Convert( object input, string inputFileFormat,
    string outputFileFormat )
{
// Todo!
}
```

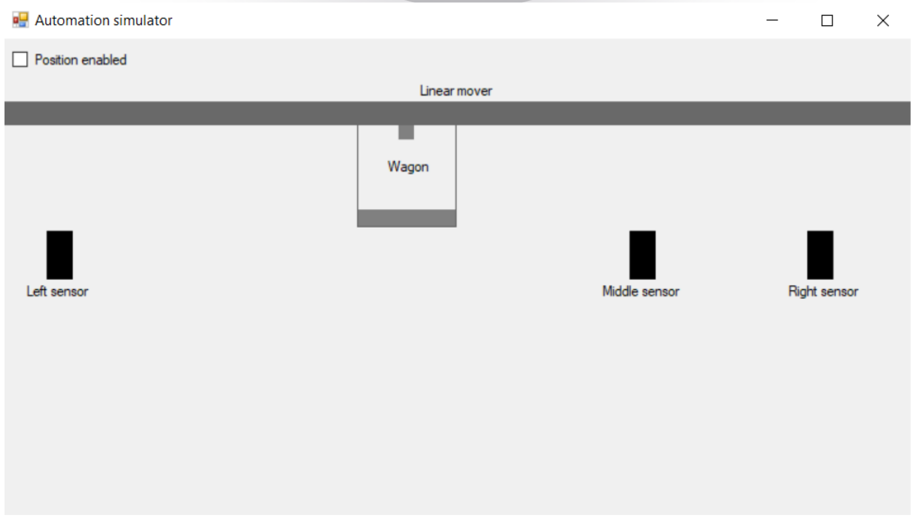
The method should find the required converters and call their `Convert` method to perform the requested conversion. Using the example above, given two converters png → jpg and jpg → bmp, we could perform a conversion png → bmp by chaining the two converters. If this is possible in more than one way, the conversion requiring less steps is preferred.

In case the requested conversion is not possible using the converters known to the manager, an appropriate exception should be thrown.

5 Automation engineering

Control an automation system according to an informal specification.

5.1 Open the solution 05_AutomationSimulator



The project consists of a physical 2D process simulation and a visualization. The components are

- There is a wagon that is moving left and right. It has a limited acceleration a which is given through the configuration.
- There are three proximity sensors: Left, middle and right. The proximity sensor triggers if the wagon is (at least partially) above it. The sensors are simulated to be ideal, i.e. they trigger immediately after detecting an object.
  - The proximity sensors are shown green when they are triggered.
- The user of the system can enable or disable positioning using a switch (which is shown as the checkbox on the top left of the screen).

The simulation and visualization of the process are already implemented, but currently, the wagon will not move. The missing part is the `Controller` class, which has one method `Update()` and is intended to control the movement of the wagon:

```csharp
public class Controller
{
    public static Outputs Update(Inputs inputs)
    {
        // TODO!
    }
}
```

The other relevant classes are `Inputs`, `Outputs` and `Configuration` defined in the Contracts.dll. The exact description of these classes and their members is written as comments in the code. Please familiarize yourself with them before continuing.

The rest of the code is only for simulation purposes and should be treated as a black box!

5.2 Design a finite state machine for the wagon control

The intended behaviour of the system is given as the following informal specification:

1. General: If the *position enabled* switch is disabled, the wagon should not move at all.
2. General: The wagon should never move past the left and right proximity sensors.
3. When the *position enabled* switch is enabled, the wagon should start moving fast towards the right until the middle proximity sensor is triggered. After that, it should continue moving slowly to the right until the right proximity sensor is reached.
4. Once the right proximity sensor is reached, the wagon should stop and position itself precisely in the center point between the middle and right proximity sensors - *as precisely as possible.* Since no information about the current speed and position of the wagon is known, the positioning needs to be timing-based.

Design a finite state machine for the wagon control using only the variables defined in the three classes `Inputs`, `Outputs` and `Configuration`.

5.3 Implement Controller.Update()

Please make changes only to the class Controller. Please treat the rest of the simulation as black box. Do not retrieve the locations or sizes of the controls from the main window, for example.