# High-Quality Code Exam – Phonebook – Evaluation Criteria

This document provides detailed evaluation criteria for the "Phonebook" exam from the "High-Quality Code" course.

## Problem 1. Code Refactoring

* **Name well the identifiers** in the code (0..4)
  + Bad named identifiers: ConsoleApplication1, Problem\_2, Class2, code, data, REP, REPNew, input (should be output), i, j, k, s, strings, str0, str1, conv, flag, Class1, Class2, sb, name2, Strings, old, obj, obj2, num, nums, oldent, newent, list, sorted, dict, multidict, found, ...
* **Format the code** well(0..3)
* Remove unused using statements: System.Data.\* and System.Net.\* (0..1)
* Removed or corrected all meaningless / wrong **comments**:
  + // this works! (0..1)
  + // Error reading from console (0..1)
  + // first command, // second command, // third command (0..2)
* Remove the unneeded **namespaces**, at least one of them (0..1)
* Fixed **incorrect error handling**:
  + Appropriate exception is thrown instead of the code “Console.WriteLine("error!"); Environment.Exit(0);” (0..2)
  + Changed the incorrect recursive call “Main();” with throwing an exception (0..2)
  + A correct exception with correct message is thrown instead of “throw new StackOverflowException();” (0..2)
  + The line "Console.WriteLine("Duplicated name in the phonebook found: " + name);" changed to throw an exception (0..2)
  + The line "Console.WriteLine("Invalid start index or count.");" changed to throw an exception (0..2)
* **Long methods** are split into small, simple and well named sub-methods:
  + Method Main() is split into several simpler methods (0..2)
  + Split the method Cmd(…) into 3 methods with better **cohesion** and less parameters: for adding a phone, for changing a phone, for listing phones (0..4)
* Simplified **complex expressions**:
  + Split the complex expression into few meaningful sub-expressions: Print("" + data.ChangePhone(convert(strings[0]), convert(strings[1])) + " numbers changed"); (0..3)
  + Split the complex expression into few meaningful sub-expressions: IEnumerable<Class1> entries = data.ListEntries(int.Parse(strings[0]), int.Parse(strings[1])); (0..2)
* **Make public** the major interfaces and classes:
  + The interface IPhonebookRepository, the classes REP and REPNew implementing the IPhonebookRepository interface, the class implementing the phonebook entry, the main program class (0..4)
* Move each public class / interface into **separate C# file** (0..1)
* Refactor the **public fields**:
  + public SortedSet<string> Strings; 🡪 public SortedSet<string> Strings {get; private set; } + initialize it in the class constructor (0..3)
  + public List<Class1> entries = new List<Class1>(); 🡪 private List<Class1> entries = new List<Class1>(); (0..2)

**45 points**

## Problem 2. Bug Fixing

* Change "**Phone entry created.**" to "**Phone entry created**" with no dot at the end (0..1)
* The code for processing the commands “**AddPhone**” and “**List**” is interchanged (0..1)
* The **command** **“**ChangeРhone**”** is wrongly spelled with Cyrillic „Р“ instead of Latin letter “P” (0..3)

**10 points**

## Problem 3. Unit Testing

* Method IPhonebookRepository.AddPhone() tested:
  + Add а **single phonebook entry** and assert it is correctly entered in the repository as new entry (0..2)
  + Add **duplicated entry** (add several times the same name + phones) and assert the entry is correctly entered in the repository only once. The first call to AddPhone() should return true, while the next calls should return false. (0..4)
  + Add multiple phones for the same name passed with **different casing** (e.g. “AddPhone(Nakov, +359111)”, then “AddPhone(naKOV, +359222)”) and assert only one phonebook entry is created. The first call to AddPhone() should return true, while the next calls should return false. (0..3)
  + Test **add with merge**: add several sets of phones for the same name and assert that in the repository the set of phones are intersected (all unique phones are kept while all duplicates are removed). (0..4)

Note that testing the whether actual content added is correct and whether the added entries are correctly sorted is not required. It will be tested through the ListEntries(…) tests.

* Method IPhonebookRepository.ListEntries(…) tested:
  + **Single entry** with single phone / multiple phones: test adding a single entry with a single phone and / or multiple phones and list it, then check for correctness (0..2)
  + **Test sorting the phonebook entries**: add several entries, list them and assert their sorting order is correct (0..3)
  + **Test sorting the phones within an entry**: add an entry with several phones, list them and assert the phones are sorted correctly (0..2)
  + **Merge + list**: add few phonebook entries with the same name to cause their phones to be merged. Then list the added entries and assert the merge was correct (0..3)
  + **Paging**: add **n** entries, then **list less than n entries** and assert the extracted page is correct. E.g. if we have 5 entries, try List(1, 3) and assert the result is correct (0..4)
  + **List invalid range (negative start)**: try to list an invalid range (negative start) and assert an exception is thrown (0..1)
  + **List invalid range (invalid start)**: try to list an invalid range (too big start) and assert an exception is thrown (0..1)
  + **List invalid range (invalid count)**: try to list an invalid range (too big count) and assert an exception is thrown (0..1)
* Method IPhonebookRepository.ChangePhone(…) tested:
  + **Change existing phone**: add a phonebook entry, then change its phone, assert 1 phone number is affected and list the phone and check it is correct (0..3)
  + **Change non-existing phone**: change a non-existing phone and assert the number of affected phones is 0 and no changes were made in the phonebook repository (0..2)
  + **Change shared phone**: add several phonebook entries that have a common (shared) phone number; change the common phone number; assert the number of affected phones is correct; list the phones and assert they are correct (0..4)
  + **Change phone with merge and duplicates**: add few phonebook entries; change a phone number with another phone number already in the phonebook to cause duplicated phones (obtained after the change) to be merged; list the phonebook and assert the phones are correctly merged (0..3)
* Check whether the **code coverage** for the class implementing IPhonebookRepository:
  + 0% ... 24% 🡪 0 points
  + 25% ... 54% 🡪 1 points
  + 55% ... 84% 🡪 2 points
  + 85% ... 100% 🡪 3 points

**40 points**

## Problem 4. Code Documentation

* No documentation 🡪 0
* Some documentation exists, but far away from finished 🡪 1
* Some interface methods documented partially 🡪 2
* Most interface methods documented partially 🡪 3
* All interface methods documented partially 🡪 4
* All interface methods documented well 🡪 5

**5 points**

## Problem 5. Performance Bottlenecks

The **first** **performance bottleneck** is that the program uses the slower IPhonebookRepository implementation: REPNew instead of REP. The implementation **based on list** (REPNew) is many times **slower** than the implementation **based on ordered sets and dictionaries (**REP**)**.

* The **fix** is very simple: just use the faster implementation class and delete the slower.
* To **test the performance** of the phonebook use a large amount of commands (e.g. 5000) or the following unit test:

|  |
| --- |
| [TestMethod]  [Timeout(3000)]  public void TestPerformance()  {  StringBuilder input = new StringBuilder();  int addCommandsCount = 1500;  for (int i = 0; i < addCommandsCount / 2; i++)  {  input.AppendLine("AddPhone(Nakov, +359887333" + i + ", +359887333999)");  input.AppendLine("AddPhone(Nakov" + i + ", +359887333999, +359887" + i + ")");  }  int changePhoneCommandsCount = 100;  for (int i = 0; i < changePhoneCommandsCount / 3; i++)  {  input.AppendLine("ChangePhone(+359887333" + (i + 100) +  ", +359887333" + (i + 200) +")");  input.AppendLine("ChangePhone(+359887333999" + ", +359887333777)");  input.AppendLine("ChangePhone(+359887333777" + ", +359887333999)");  input.AppendLine("ChangePhone(+359887" + (i + 500) + ", +359887333999)");  }  int listCommandsCount = 5000;  for (int i = 0; i < listCommandsCount; i++)  {  input.AppendLine("List(" + i + ", " + (1 + (i % 20)) + ")");  }  input.AppendLine("End");  // Forcefully invoke the static constructor to ensure the program state is clean  typeof(PhonebookSystem.Phonebook).TypeInitializer.Invoke(null, null);    // Redirect the console input / output and invoke the Main() method  Console.SetIn(new StringReader(input.ToString()));  StringWriter consoleOutput = new StringWriter();  Console.SetOut(consoleOutput);  PhonebookSystem.Phonebook.Main();  // Assert is not needed, just check the performance  // Assert.AreEqual(expected, actual);  } |

Evaluation criteria:

* The performance bottleneck is found (e.g. marked by a comment or fixed) 🡪 2
* The performance bottleneck is fixed correctly (the unit test passes) 🡪 1

The **second** **performance bottleneck** is in the method “conv(…)” that converts a phone number into canonical form. It has **unneeded** for**-loop** that does nothing but repeating the phonebook conversion logic many times. In the loop body the **conversion is made several times** with no good reason.

* The **fix** is easy: remove the unneeded for-loop and repeated conversion logic.
* To **test the performance** of the conv(…) method you may use the following unit test:

|  |
| --- |
| [TestMethod]  [Timeout(1000)]  public void TestPerformanceConvertPhoneToCannonicalForm()  {  // Prepare input commands  StringBuilder input = new StringBuilder();  int addCommandsCount = 1000;  for (int i = 0; i < addCommandsCount; i++)  {  input.AppendLine("AddPhone(Nakov, 02 / 981 45 66)");  input.AppendLine("AddPhone(Nakov, +359 899 99 22 22)");  input.AppendLine("AddPhone(Nakov, (062) 62 62 62)");  input.AppendLine("AddPhone(Nakov, (+359) 899 777-555)");  }  input.AppendLine("List(0, 1)");  input.AppendLine("End");  // Prepare the expected result  StringBuilder expectedOutput = new StringBuilder();  expectedOutput.AppendLine("Phone entry created");  for (int i = 0; i < 4 \* addCommandsCount - 1; i++)  {  expectedOutput.AppendLine("Phone entry merged");  }  expectedOutput.AppendLine(  "[Nakov: +35929814566, +35962626262, +359899777555, +359899992222]");  // Forcefully invoke the static constructor to ensure the program state is clean  typeof(PhonebookSystem.Phonebook).TypeInitializer.Invoke(null, null);  // Redirect the console input / output and invoke the Main() method  Console.SetIn(new StringReader(input.ToString()));  StringWriter consoleOutput = new StringWriter();  Console.SetOut(consoleOutput);  PhonebookSystem.Phonebook.Main();  // Assert that the program execution result is correct  string expected = expectedOutput.ToString();  string actual = consoleOutput.ToString();  Assert.AreEqual(expected.Length, actual.Length);  } |

Evaluation criteria:

* The performance bottleneck is found (e.g. marked by a comment or fixed) 🡪 2
* The performance bottleneck is fixed correctly (the unit test passes) 🡪 2

The **third** **performance bottleneck** is in the method ChangePhone() of IPhonebookRepository‘s fast implementation. It performs ToList() for all matched phonebook entries which is slow operation. No fix is available.

* To **test the performance** of the ChangePhone() method use a large amount of ChangePhone commands (e.g. 1000) over enough matching phonebook entries (e.g. 500) or the following unit test:

|  |
| --- |
| [TestMethod]  [Timeout(1000)]  public void TestPerformanceChangePhone()  {  // This test is expected to fail due to timeout. It just shows that  // a performance bottleneck exists in the ChangePhone() method.  StringBuilder input = new StringBuilder();  int addCommandsCount = 2000;  for (int i = 0; i < addCommandsCount; i++)  {  input.AppendLine("AddPhone(Nakov" + i + ", +359887333999)");  }  int changePhoneCommandsCount = 500;  for (int i = 0; i < changePhoneCommandsCount / 2; i++)  {  input.AppendLine("ChangePhone(+359887333999" + ", +359887333777)");  input.AppendLine("ChangePhone(+359887333777" + ", +359887333999)");  }  input.AppendLine("End");  // Forcefully invoke the static constructor to ensure the program state is clean  typeof(PhonebookSystem.Phonebook).TypeInitializer.Invoke(null, null);  // Redirect the console input / output and invoke the Main() method  Console.SetIn(new StringReader(input.ToString()));  StringWriter consoleOutput = new StringWriter();  Console.SetOut(consoleOutput);  PhonebookSystem.Phonebook.Main();  // Assert is not needed, just check the performance  } |

Evaluation criteria:

* The performance bottleneck is found and described in the code 🡪 3

**Bonus: up to 10 points**