ReSharper Course Exercises

First Exercise

In the **Basic** solution. Create the 2 following classes:

- **Plan**
- **Document**

The **Plan** class should take a name (**string**), some documents (**IEnumerable<Document>**) and a security classification (**Basic.Support.SecurityClassification**) in it's constructor and expose them as read-only properties.

One solution could look like this:

Plan.cs

```
using System.Collections.Generic;
using Basic.Support;

namespace Basic
{
    public class Plan
    {
        private readonly string _name;
        private readonly IEnumerable<Document> _documents;
        private readonly SecurityClassification _securityClassification;

        public Plan(string name, IEnumerable<Document> documents, SecurityClassification securityClassi
        {
            _name = name;
            _documents = documents;
            _securityClassification = securityClassification;
        }

        public string Name
        {
            get { return _name; }
        }

        public IEnumerable<Document> Documents
        {
            get { return _documents; }
        }

        public SecurityClassification SecurityClassification
        {
            get { return _securityClassification; }
        }
    }
}
```

Document.cs

```
namespace Basic
{
    public class Document
    {
    }
}
```

Primary shortcuts

Focus Solution Explorer	Alt + Ctrl + L
Locate in Solution Explorer	Alt + Shift + L
Open context menu	Shift + F10
Focus Code Window	ESC
Focus Code Window (more stable)	Alt + W , 1
Open menu underlined with <X>	Alt + <X>

Go to ...

Find the shortest sequence of keys to navigate to following items. See example below.

Example - Shortest Sequence

The shortest sequence of keys to navigate to

- The file **`templates/roles.htm`**

would be:

- Ctrl + Shift + T (Go to file)
- r h t (matches roles.htm)
- Enter

1. The class **`IActivateHandlers`** interface in the **`Rebus`** namespace
2. The class **`QuickNotes`** class in the **`BlogEngine.Core.Notes`** namespace
3. The file **`blog.js`** file in the **`Scripts/Header/`** folder
4. The class **`BlogReader`** in the **`BlogEngine.Core.API.BlogML`** namespace
5. The declaration of the **`LoginRequired`** css class in the **`Styles/Global.css`** file
6. The **`Page_Load`** method in the **`Account/login.aspx.cs`** file
7. The **`staticContent`** xml section in the **`web.config`** file
8. The **`Application_Error`** method in the **`Global.asax`** file
9. The **`cancelReply`** method in the **`Scripts/Header/blog.js`** file
10. The **`Add`** method of the **`BlogEngine.Core.Providers.BlogFileSystemProviderCollection`** class
11. The class that starts with **`U`** in the same namespace as **`XmlFileSystemProvider`**

Primary shortcuts

Go to Type	Ctrl + T
Go to File	Ctrl + Shift + T
Go to Symbol	Alt + Shift + T
Go to Member (current file)	Ctrl + <

Supporting shortcuts

Locate in Solution Explorer	Alt + Shift + L

Code Analysis

Navigate to the **`ContextClass`** in the **`CodeAnalysis`** solution.

The class contains various ReSharper issues.

- **Hints** shown as a short dotted green line beneath the code
- **Suggestions** shown as a green squiggly line beneath the code
- **Dead code** shown as faded grey text
- **Warnings** shown as a blue squiggly line beneath the code
- **Errors** shown as red text or red squiggly lines

Do the following

1. Navigate to the 2 errors and fix them. What's wrong?
2. Navigate through the suggestions in the file and see what ReSharper is suggesting. Do you agree?
3. Go through the hints of the file (no shortcut). Does it make sense to apply any of them?

> **Note:** The inspection options can be changed either in the ReSharper options or directly through the Quick fix menu. These can also be shared at solution level through source control with the new ReSharper 6.x collaboration features.

Primary shortcuts

Quick fix	Alt + Enter
Go to next suggestion	Alt + PageDown
Go to previous suggestion	Alt + PageUp
Go to next error	Alt + Shift + PageDown
Go to previous error	Alt + Shift + PageUp

Generate Code

Open the **`GenerateCode`** class in the **`Basic`** project.

1. From the constructor of **`GenerateCode`** call a the non-existant private method like so: **`Create(name, tags)`**
2. Use Quick fix to create the method
3. From the **`Create`** method instantiate a non-existant class like so: **`new CodeCollection(name, tags)`**
4. Use Quick fix to create the class + constructor
5. Assign the parameters of the **`CodeCollection`** to fields using Quick fix
6. Delete the constructor
7. Use Generate code to re-create the constructor
8. Use Generate code to create a read-only property for the **`name`** field
9. Generate a property for the **`tags`** field
10. Generate equality members for the class depending on the **`name`** field
11. Generate delegating methods for the **`Add`** method and the **`Count`** properties of the **`tags`** list

Primary shortcuts

Quick fix	Alt + Enter
Generate code	Alt + Ins

Supporting shortcuts

Go to next suggestion	Alt + PageDown
Go to previous suggestion	Alt + PageUp

Rename

Navigate to the `H` class the `Basic` project.

1. Rename variable `e` -> `lastIndex`
2. Rename parameter `j` -> `arrayLength`
3. Rename method `A` -> `BuildRandomArray`
4. Rename variable `d` -> `i`

Figure out what variables (and rename them) should have the following names of the remaining `c` , `H` , `g` and `f` .

- `random`
- `numbers`
- `swapIndex`
- `temp`
- `ArrayShuffler`

Primary shortcuts

Refactor: Rename	Ctrl + R , Ctrl + R

Supporting shortcuts

Refactor this	Ctrl + Shift + R
Highlight usages	Alt + Shift + F11

Introduce and Inline variables

Open the **`Variables`** class in the **`Basic`** project.

In the **`IntroduceVariable`** method introduce the following variables

1. `0` to **`firstPage`**
2. **`numberOfPages - 1`** to **`lastPage`**
3. **`Math.Min(Math.Max(firstPage, page), lastPage)`** to **`currentPage`**
4. **`currentPage * itemsPerPage`** to **`firstItemInPage`**
5. **`((currentPage + 1) * itemsPerPage) - 1`** to **`lastItemInPage`**
6. **`new[] { firstItemInPage, lastItemInPage }`** to **`itemRange`**

In the **`InlineVariable`** method:

7. Inline all the variables one at a time and watch the effects.

Primary shortcuts

Refactor this	Ctrl + Shift + R
Refactor: Introduce variable	Ctrl + R , Ctrl + V
Refactor: Inline variable	Ctrl + R , Ctrl + I

Import Completion

Open the `ImportCompletion` class in the `Basic` project.

In the `ImportCompletion` method:

1. Instantiate a `new DataProcessor()` using Import completion to import the `Basic.Support.Proc` namespace
2. Change the `Console.WriteLine` call to output `list.FirstOrDefault()` (extension method) instead of `list` - using Import completion to import the `System.Linq` namespace
3. Instantiate a `System.Threading.Timeout`
4. Instantiate a `System.Security.SecureString`

In the `IntroduceVariableImportCompletionCombo` method, find the shortest keysequence for writing the following statements using Import completion followed by Introduce variable . (see example below)

5. `var stringBuilder = new StringBuilder();`
6. `var dictionary = new HybridDictionary();`
7. `var collection = new BlockingCollection<ConcurrentQueue<Guid>>();`
8. `var compressionMode = CompressionMode.Compress;`

Example - Import/Introduce combo

Combining Import completion and Introduce variable . Shortest key sequence for writing in a file where the `System.Text` namespace is not imported:

- `var stringBuilder = new StringBuilder();`

would be:

- n e w Space
- s b u i (matches **S**tring**Bui**lder)
- Alt + Shift + Space (Activate Import completion)
- ((inserts auto-closing paren)
- Ctrl + R , Ctrl + V (introduce variable)
- Enter (picks *var* in type selector)
- Enter (picks *stringBuilder* in name selector)

Primary shortcuts

Refactor: Introduce variable	Ctrl + R , Ctrl + V
Import completion	Alt + Shift + Space
Refactor this	Ctrl + Shift + R

Find Usages and Highlight

Investigate the following using Find usages

1. Where is the **`DumpProvider`** method from the **`FileSystemUtilities`** class used?
2. Which .aspx files use the **`CheckRightsForAdminCommentsPages`** method in the **`WebUtils`** class?
3. From which .aspx files can the **`SaveToDatastore`** method of the **`XmlBlogProvider`** class be invoked? (multiple Find usages)

Navigate to the **`BlogEngine.Core.Web.Navigation.Pager`** class and look at the **`Pager`** constructor (note: there is more than 1 Pager class)

4. Try to reason about what happens to the **`page`** parameter within the constructor
5. Try again with Find usages
6. Try again with Highlight usages
7. What works better? Nothing, Find usages or Highlight usages ?

Navigate to the **`ContextBeginRequest`** method in the **`UrlRewrite`** class. Using Highlight usages or Find usages figure out

8. What is the last line that the **`path`** local variable is used in?
9. What is the last line that the **`url`** local variable is used in?

Navigate to the **`BlogEngine.Core.Web.Navigation.Pager`** class

10. Try to reason about who calls the **`Pager`** constructor
11. Try again with Highlight usages
12. Try again with Find usages
13. What works better? Nothing, Find usages or Highlight usages ?

Primary shortcuts

Find usages	Shift + F12
Highlight usages	Alt + Shift + F11
Remove Highlight	Esc

Supporting shortcuts

Go to next usage	Alt + Ctrl + PageDown
Go to previous usage	Alt + Ctrl + PageUp

Solution Explorer Refactorings

Open the **Tennis.cs** file in the **Tennis** solution. This file contains an implementation of the variations of the rules of Tennis in different tournaments.

In the Solution Explorer do the following:

1. Split up the **Tennis.cs** file using Refactor this - Move types into matching files
2. Create **Tournament** and **Models** folders using Generate file (Solution Explorer)
3. Move files into the folder structure shown below using either
   - Cut / Paste + Refactor this - Adjust namespaces
   - Refactor this - Move to folder
4. Which method do you like better?

- **Tournament**
  - **TournamentAustralianOpen.cs**
  - **TournamentDouble.cs**
  - **TournamentUSOpen.cs**
  - **ITournamentRules.cs**
- **Models**
  - **TennisGame.cs**
  - **TennisSet.cs**
  - **TennisMatch.cs**

> **Tip:** If you want to start over on this task - simply copy the contents of **original.txt** into a **Tennis.cs** file.

Primary shortcuts

Refactor this	Ctrl + Shift + R
Generate file (Solution Explorer)	Alt + Ins
Refactor: Move to folder (Solution Explorer)	Ctrl + R , Ctrl + O

Supporting shortcuts

Locate in Solution Explorer	Alt + Shift + L

Move Code

Navigate to the `OutOfOrderMethods` class in the `Basic` project.

1. Reorder the methods alphabetically using move code.

In the `E` method:

2. Move the calculation of `theOtherValue` into the first `if` block.
3. Move the `try` / `catch` block out of the first `if` block.
4. Change following expressions to mention variable names before constants:
   - `13 * theNumber` should be `theNumber * 13`
   - `42 == theNumber` should be `theNumber == 42`
   - `null != data` should be `data != null`

Primary shortcuts

Move code up	Alt + Ctrl + Shift + Up
Move code down	Alt + Ctrl + Shift + Down
Move code in	Alt + Ctrl + Shift + Right
Move code out	Alt + Ctrl + Shift + Left

Supporting shortcuts

Go to next class member	Alt + Down
Go to previous class member	Alt + Up

Navigate Hierarchies

Navigate to the **`Navigate`** method of the **`NavigateHierachies`** class.

1. Examine the Type Hierarchy of **`IEntity`**
2. What is the difference between Go to implementation and Go to derived on the **`IPet.Speak`** method?
3. Where does Go to implementation on **`IEntity.Id`** followed by Go to base take you? Why?

Navigate to the **`RebusHierarchyLesson`** class. Using hierarchy navigation (Go to declaration / Go to implementation / Go to base / Go to derived) figure out

Remember that you can consult the Type Hierachy .

4. From **`SimpleHandlerActivator`** - how can you get to **`IActivateHandlers`**
5. From **`IActivateHandlers`** - how can you get to **`WindsorContainerAdapter`**
6. From **`WindsorContainerAdapter`** - how can you get to **`IContainerAdapter`**

Without leaving **`RebusHierarchyLesson`** figure out the following from the **`Lesson`** method.

> **Hint:** Go to derived shows a list of possible navigation targets for whatever's under the caret.

7. Using Type Hierachy, which class implementing **`IActivateHandlers`** does not implement **`IContainerAdapter`**
8. Number of classes and interfaces implementing **`IActiveHandlers`**
9. Out of the these - how many implement the **`GetHandlerInstancesFor<T>`**
10. Number of classes and interfaces implementing **`IHandleMessages<T>`**

> **Tip:** If you navigate to a file and want to go back to where you were, you can either use Close current file or Go back (Visual Studio) .

Primary shortcuts

Go to declaration	F12
Go to implementation	Ctrl + F12
Go to base	Alt + Home
Go to derived	Alt + End
Show type hierarchy	Ctrl + E , Ctrl + H

Supporting shortcuts

Inspect this	Alt + Ctrl + Shift + A
Close current file	Ctrl + F4
Go back (Visual Studio)	Ctrl + -

Inspect This

Navigate to the **`Inspector`** class in the **`Basic`** project.

> **Tip:** Remember that the inspection window can be docked and that Show Preview on the Right is useful.

1. Explore the value destination of **`importantValue`** in the **`Main`** method. Where does it end up?
2. Explore the value origin of the **`value`** variable in the **`Main`** method. What values are printed?

Primary shortcuts

Inspect this	Alt + Ctrl + Shift + A

First Exercise Revisited

Repeat the first exercise, but try to incorporate the things you've learned in the previous exercises. Below you'll find a list of the shortcuts that might be helpful for faster flows.

Primary shortcuts

Quick fix	Alt + Enter
Generate file (Solution Explorer)	Alt + Ins
Generate code	Alt + Ins
Import completion	Alt + Shift + Space
Go to next suggestion	Alt + PageDown
Go to previous suggestion	Alt + PageUp
Go to next error	Alt + Shift + PageDown
Go to previous error	Alt + Shift + PageUp