EECS3311 Fall 2017
Lab Exercise 0
Practicing Design-by-Contract in Eiffel Studio

Chen-Wei Wang

Abstract

This lab is intended to help you get familiar with the programming/design environment of this course. We will learn about the most common features of Eiffel Studio (an IDE for the object-oriented Eiffel programming language, analogous to Eclipse for Java). We will also learn about drawing design diagrams that conform to the BON (Business Oriented Notation) standard, where you present: **1)** the architecture of your class design; and **2)** the specification (i.e., contract) of features in those classes.

It is recommended that you complete this lab exercise using your EECS accounts (e.g., from machines located in LAS1006), so as to prepare yourself for the lab tests which will take place in the same working environment.

Screen shots of this lab are taken from a Mac OS X environment, but the look and feel of Eiffel Studio on your lab accounts should be fairly similar.

There is no submission required for this Lab 0, but you should complete it as soon as possible, no later than the scheduled date of Lab 1.

Contents

1 Acronyms

- **EStudio** [Eiffel Studio]
- **DbC** [Design by Contract]
- **TDD** [Test-Driven Development]

2 Create a Workspace for Your Eiffel Projects

- Launch a terminal (right click on your desktop, then there should be an option for that).
- Type the following command to: **1)** change the current director to your desktop; and **2)** create a workspace for your Eiffel lab projects.

```
cd ~/Desktop
mkdir EECS3311_Labs
mkdir EECS3311_Labs/Lab_0
```

where the symbol ~ is a shorthand for the path of your home directory (e.g., `/eecs/home/jackie`).

Now, you have on your desktop a directory `EECS3311_Labs`, which contains only an empty directory `Lab_0`.

3 Launching Eiffel Studio

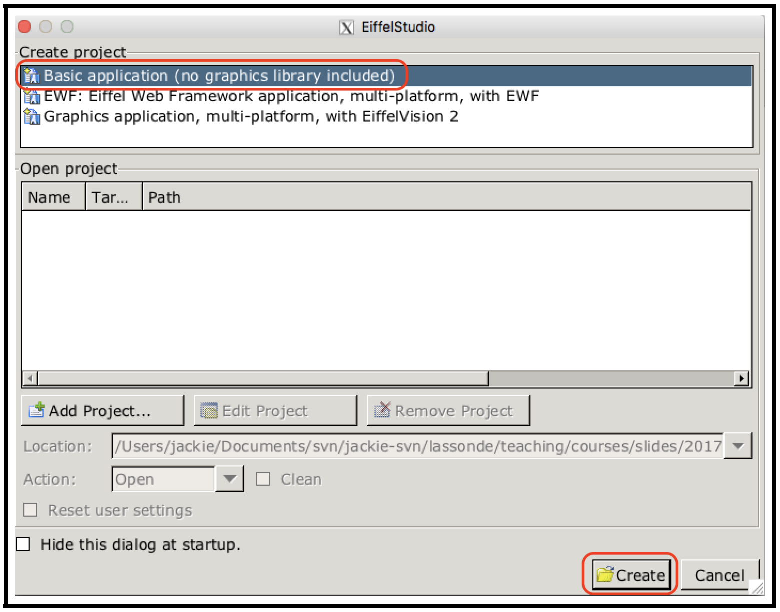
- On your terminal, type the following command to launch the latest version of Eiffel Studio (17.05)

```
estuio17.05 &
```

where the `&` means, as you learned from your EECS2031, that the process will be executed at the background of the current terminal, whereas you do not need to open another terminal to execute other commands if needed.

4 Creating a Project

- Right after EStudio is launched, a window will pop up:



EiffelStudio

Create project

- Basic application (no graphics library included)
- EWF: Eiffel Web Framework application, multi-platform, with EWF
- Graphics application, multi-platform, with EiffelVision 2

Open project

Name	Tar...	Path

Location: /Users/jackie/Documents/svn/jackie-svn/lassonde/teaching/courses/slides/2017

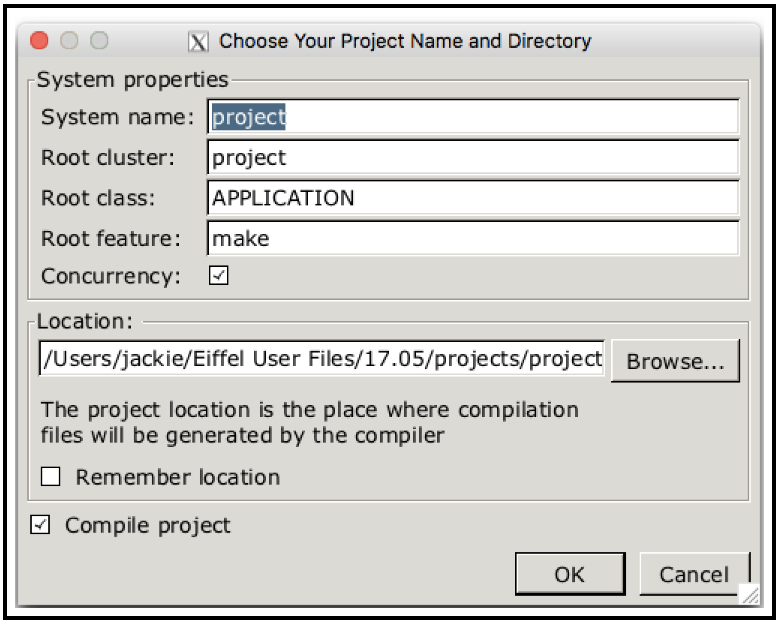
Action: Open □ Clean

□ Reset user settings

□ Hide this dialog at startup.

- Make sure under the `Create project` panel, the option `Basic application (no grphics library included)` is selected.
- Click on `Create`.
- A window will pop up for you to choose the project name and location.

  In Eiffel convention, project name consists of all lower cases, separated by underscores (_) for compound words (i.e., same as names of Java packages).



Choose Your Project Name and Directory

System properties

System name: project

Root cluster: project

Root class: APPLICATION

Root feature: make

Concurrency: ☑

Location:

/Users/jackie/Eiffel User Files/17.05/projects/project

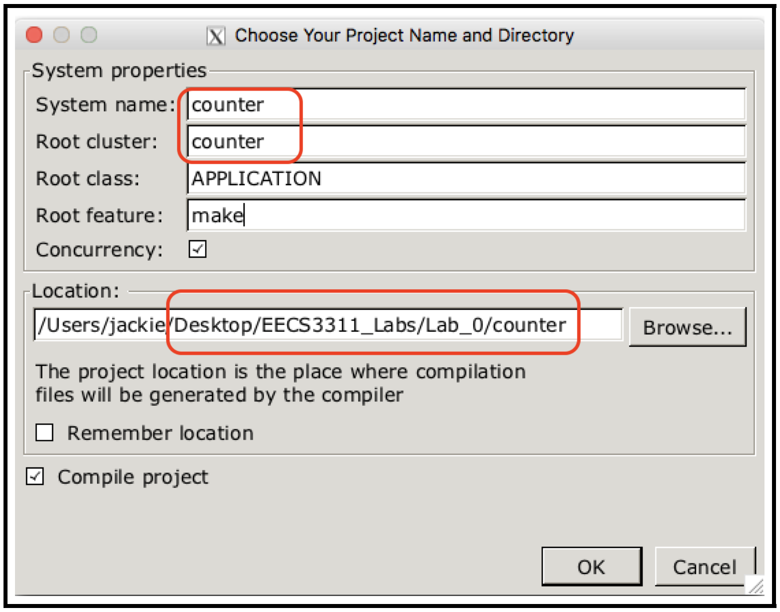
The project location is the place where compilation files will be generated by the compiler

□ Remember location

☑ Compile project

- For `System name`: change it to `counter`
- For `Root cluster`: change it to `counter`
- Leave `Root class` and `Root feature`, leave them as they are.
- For `Location`, first use `Browse` to get the path for directory `Lab_0` that you created, then append `/Lab_0` to the end of it.

  You should now see something like this:



Choose Your Project Name and Directory

System properties

System name: counter

Root cluster: counter

Root class: APPLICATION

Root feature: make

Concurrency: ☑

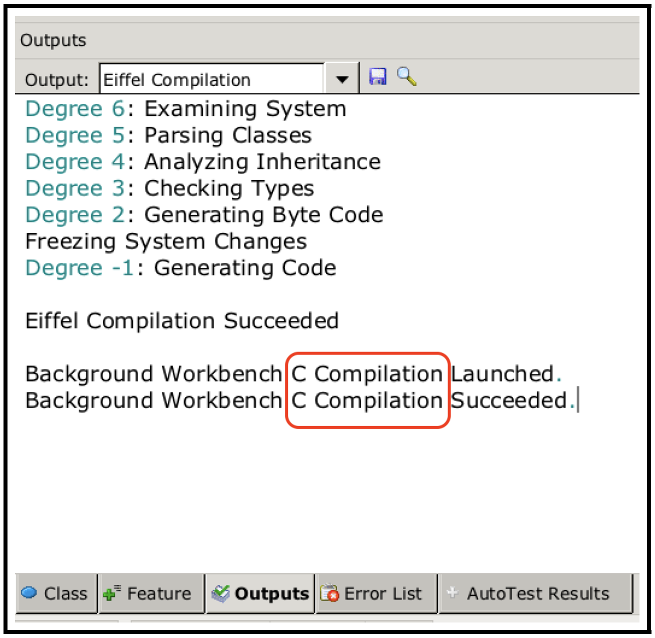
Location: /Users/jackie/Desktop/EECS3311_Labs/Lab_0/counter

The project location is the place where compilation files will be generated by the compiler

□ Remember location

☑ Compile project

◇ Click on `OK` to start compiling the project, wait until you see from the `Outputs` panel that the compilation is successfully completed.



Outputs

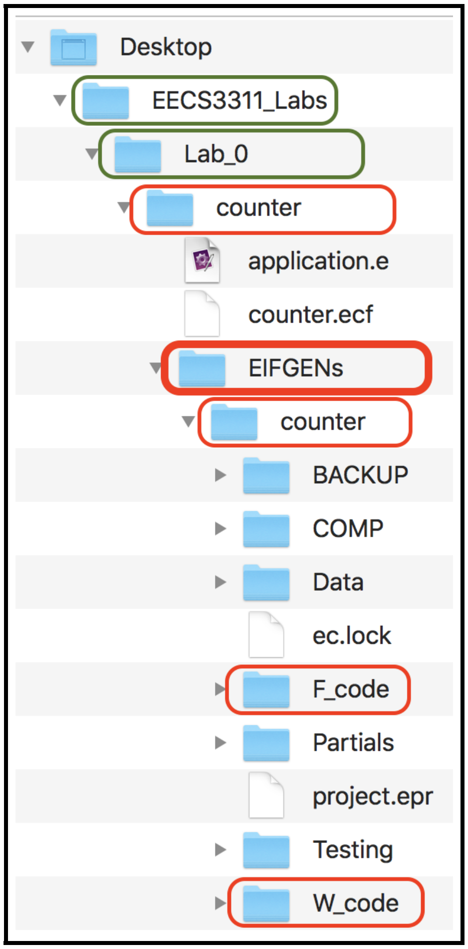
Output: Eiffel Compilation

```
Degree 6: Examining System
Degree 5: Parsing Classes
Degree 4: Analyzing Inheritance
Degree 3: Checking Types
Degree 2: Generating Byte Code
Freezing System Changes
Degree -1: Generating Code

Eiffel Compilation Succeeded

Background Workbench C Compilation Launched.
Background Workbench C Compilation Succeeded.
```

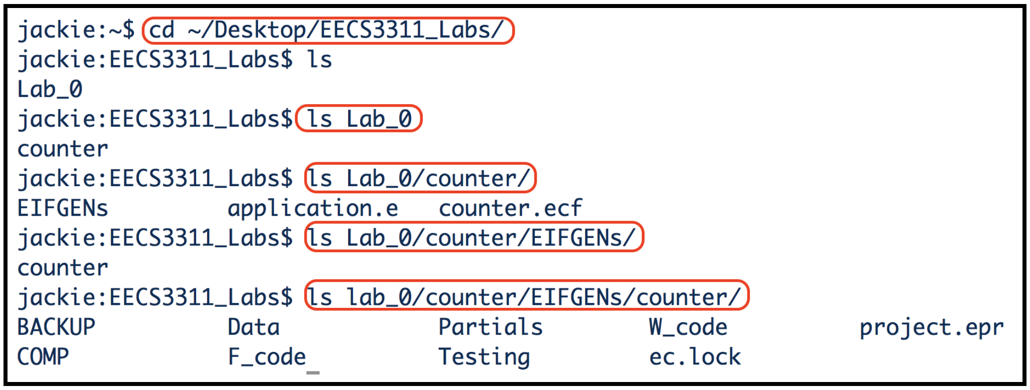
Remark: The output message above includes phrases of `C Compilation`. Keep in mind that code written in Eiffel is not directly executable; instead, they are compiled and, for efficiency reason, optimized into C code. All C code is stored in a subdirectory of your project named `EIFGENs` (standing for Eiffel Generations). We will explore this `EIFGENs` directory more later. This compilation makes sense: Eiffel is meant to be a **design language** for you to think at a much higher level of abstraction, whereas C is meant to be an **implementation language** for you to tweak the performance of your code (e.g., exploiting pointer arithmetic).

5 Exploring Project Structure from the File System

– Using the GUI-based file explorer, convince yourself that the project `counter` that you created has the following directory structure in the file system:



– Alternatively, using the terminal, reaffirm yourself about the directory structure:



```
jackie:~$ cd ~/Desktop/EECS3311_Labs/
jackie:EECS3311_Labs$ ls
Lab_0
jackie:EECS3311_Labs$ ls Lab_0
counter
jackie:EECS3311_Labs$ ls Lab_0/counter/
EIFGENs         application.e   counter.ecf
jackie:EECS3311_Labs$ ls Lab_0/counter/EIFGENs/
counter
jackie:EECS3311_Labs$ ls lab_0/counter/EIFGENs/counter/
BACKUP          Data            Partials        W_code          project.epr
COMP            F_code          Testing         ec.lock
```

Understanding the Critical Directories

- Since we chose the project location to be a subdirectory `counter` under `Lab_0`, all bits and pieces related to this current project is compiled into this subdirectory.
- The directory `EIFGENs` (standing for Eiffel Generations) stores all the C code this is compiled from the source Eiffel cod e in the current project. Since we chose the project name as `counter`, this results in the fact that there is a subdirectory named `counter` under the `EIFGENs` directory. There are two subdirectories under `EIFGENs/counter` that you should know about:

- **F_code**: This directory stores the finalized (i.e., optimized and ready for delivery/submission) executable of your project. When your new project is first created and compiled, the **default** option is that the project is **not** finalized, meaning that it is still subject to a number of intermediate revisions/recompilations. To see this, on your terminal type the following command:

```
ls ~/Desktop/EECS3311_Labs/Lab_0/counter/EIFGENs/counter/F_code
```

  You should see an empty output. Why? Because nothing has been finalized yet!

- **W_code**: As said, when your project is first created and compiled, it is still subject to a number of recompilations until you are satisfied so that you can finalize your project. Currently there is an intermediate (i.e., not optimized) executable that exists in this `W_code` directory. To see this, type the following command:

```
ls ~/Desktop/EECS3311_Labs/Lab_0/counter/EIFGENs/counter/W_code
```

  The output should be:

```
C1  Makefile     TRANSLAT   counter
E1  Makefile.SH  config.sh  counter.melted
```

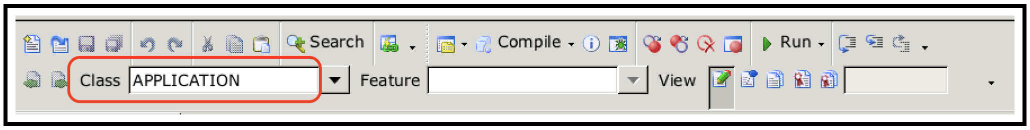
  where the file `counter` is in fact an executable. Try it by typing:

```
~/Desktop/EECS3311_Labs/Lab_0/counter/EIFGENs/counter/W_code/counter
```

  The output should be:

```
Hello Eiffel World!
```

  Now switch back to EStudio, open the class `APPLICATION` by typing on the `Class` text box on the top.



Class: APPLICATION

  You will see a line in the class that reads: `print ("Hello Eiffel World!%N")`. Notice that the percentage sign `%` there means the start of an escape sequence (whereas in Java, you start an escape sequence using a backward slash \). The Eiffel escape sequence `%N` here denotes the new-line character.

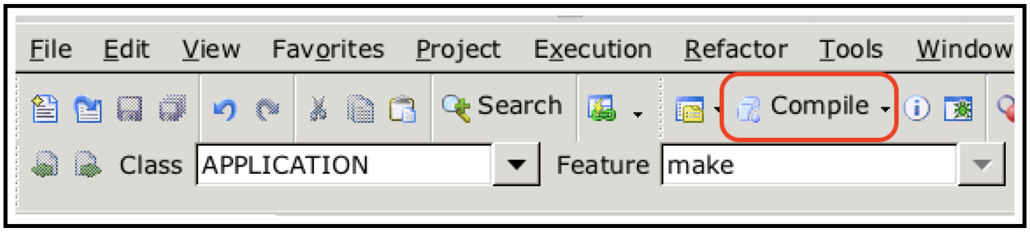
  Now, modify that line to: `print ("Hello Eiffel World @ EECS3311!%N")` and save the file (`Ctrl` + `s`). Then, switch back to your terminal and run the (un-finalized) executable again from `W_code`:

```
~/Desktop/EECS3311_Labs/Lab_0/counter/EIFGENs/counter/W_code/counter
```

  The output should be:

```
Hello Eiffel World!
```

  Aah! Should the output be changed to `Hello Eiffel World @ EECS3311!%N`?! The reason for this is because **we did not re-compile for the changed code to take effect**. We somehow have been spoiled by Eclipse in the previous Java courses, where a re-compilation is performed automatically as soon as you save your Java file. **In EStudio, saving a file does not trigger re-compilation automatically.** To fix this, re-compile your code from EStudio:



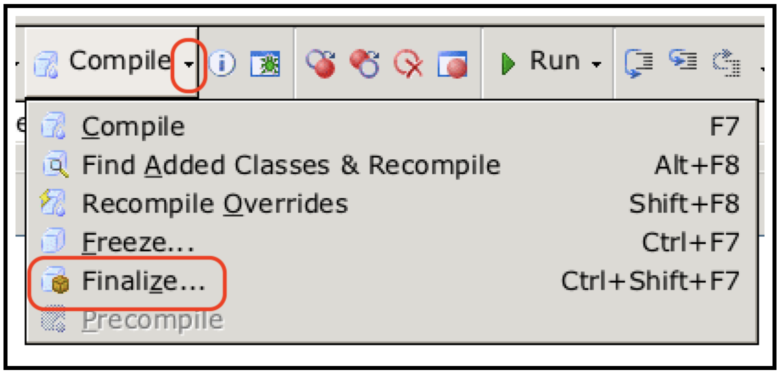
Try running the executable again:

```
~/Desktop/EECS3311_Labs/Lab_0/counter/EIFGENs/counter/W_code/counter
```

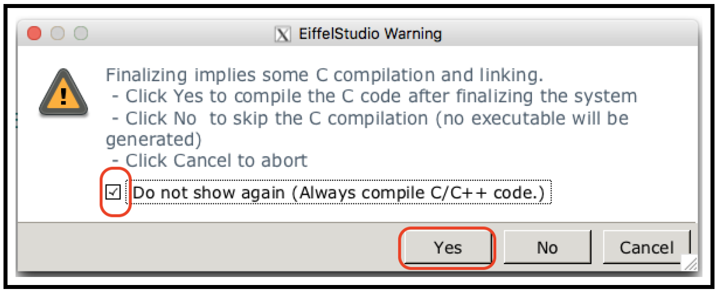
You should now get the expected output:

```
Hello Eiffel World @ EECS3311!
```

- The next question is: How do we finalize the project then? In EStudio, click on the tiny, upside-down triangle symbol right beside `Compile`, and this will give you a drop-down menu list of options: click on `Finalize...`.


Compile
Run
Compile F7
Find Added Classes & Recompile Alt+F8
Recompile Overrides Shift+F8
Freeze... Ctrl+F7
Finalize... Ctrl+Shift+F7
Precompile

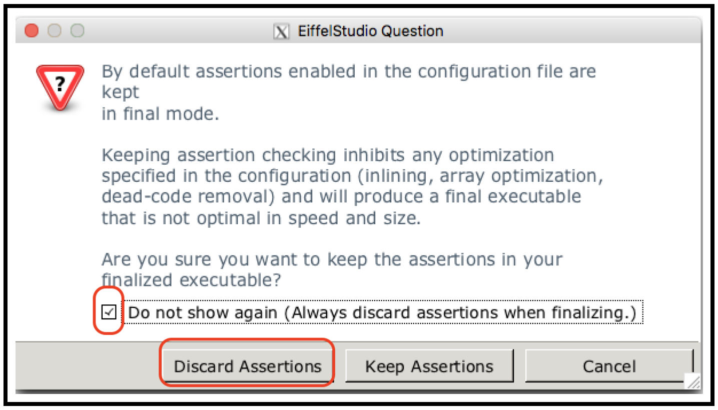
  - ◇ Tick the check box `Do not show again (Always compile C/C++ code)` and click `Yes`.



Finalizing implies some C compilation and linking.
- Click Yes to compile the C code after finalizing the system
- Click No to skip the C compilation (no executable will be generated)
- Click Cancel to abort

☑ Do not show again (Always compile C/C++ code.)

  - ◇ Tick the check box `Do not show again (Always discard assertions when finalizing)` and click `Discard Assetions`.



By default assertions enabled in the configuration file are kept
in final mode.

Keeping assertion checking inhibits any optimization specified in the configuration (inlining, array optimization, dead-code removal) and will produce a final executable that is not optimal in speed and size.

Are you sure you want to keep the assertions in your finalized executable?

☑ Do not show again (Always discard assertions when finalizing.)

Then wait until the `Outputs` panel indicates that the compilation is completed.

- After finalizing the project, let's switch back to the terminal and type:

  ```
  ls ~/Desktop/EECS3311_Labs/Lab_0/counter/EIFGENs/counter/F_code
  ```

  You will see that there is, similar to the case of `W_code`, also an executable file called `counter`. Run this (finalized, optimized) executable by typing:

  ```
  ~/Desktop/EECS3311_Labs/Lab_0/counter/EIFGENs/counter/F_code/counter
  ```

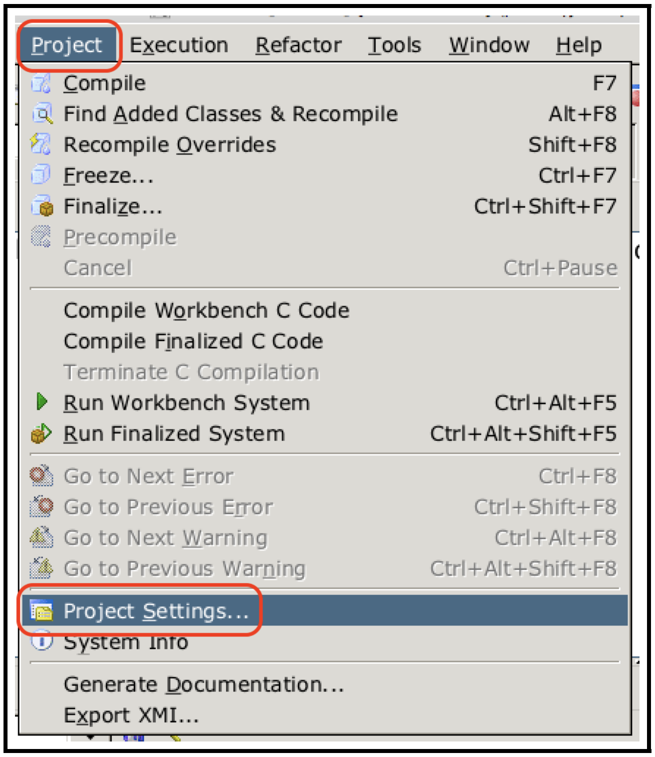
Remark: The executables in `W_code` and `F_code` have no difference in terms of their behaviour. It is only that one (in `F_code`) has better performance than the other (in `W_code`). While your still developing your project, there is no need to finalize your code, as it takes time to finalize/optimize each time.

6 Setting the Project Cluster Structure

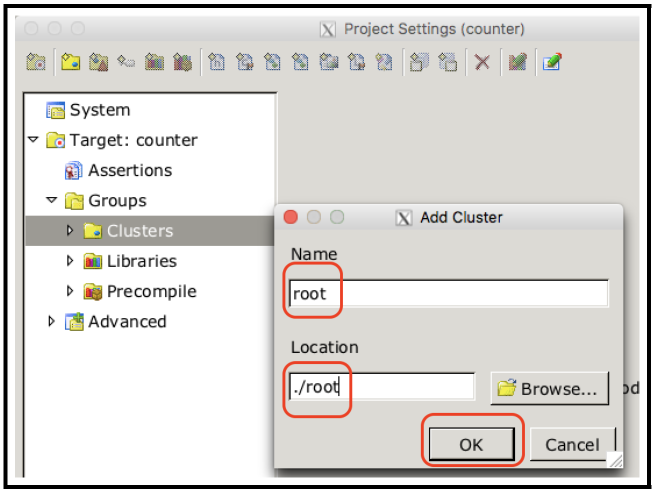
- In Eclipse, you use packages to group your Java classes. In EStudio, we use **clusters**. Each cluster (which is essentially a symbolic link) can be set to correspond to a physical directory on your file system.
- We first create the physical directories on your file system from the terminal, and then we set the corresponding links to them as clusters from EStudio.
  - Create the following three directories (right under the project location) from your terminal:

    ```
    cd ~/Desktop/EECS3311_Labs/Lab_0/counter
    mkdir root
    mv application.e root
    mkdir model
    mkdir tests
    ```

    where we also move the file `application.e` into the directory `root`. Typically:
    - The `root` directory stores Eiffel classes (whose file extension is `.e`) that controls the execution.
    - The `model` directory stores classes related to solving the problem of interests.
    - The `tests` directory stores classes related to testing your model classes.
  - After making these changes on your file system, let's now switch back to EStudio to create the clusters accordingly.
    - From the top menu list, select `Project` and then `Project Settings...`.


Project
Execution
Refactor
Tools
Window
Help
Compile F7
Find Added Classes & Recompile Alt+F8
Recompile Overrides Shift+F8
Freeze... Ctrl+F7
Finalize... Ctrl+Shift+F7
Precompile
Cancel Ctrl+Pause
Compile Workbench C Code
Compile Finalized C Code
Terminate C Compilation
Run Workbench System Ctrl+Alt+F5
Run Finalized System Ctrl+Alt+Shift+F5
Go to Next Error Ctrl+F8
Go to Previous Error Ctrl+Shift+F8
Go to Next Warning Ctrl+Alt+F8
Go to Previous Warning Ctrl+Alt+Shift+F8
Project Settings...
System Info
Generate Documentation...
Export XMI...

- ◇ A window for `Project Settings` should pop up. From its left panel, click on `Groups`, then you should see `Clusters` and `Libraries`.
- ◇ Right-click on `Clusters`, choose `Add Cluster`, then type `root` for `Name` and type `./root` for `Location`.



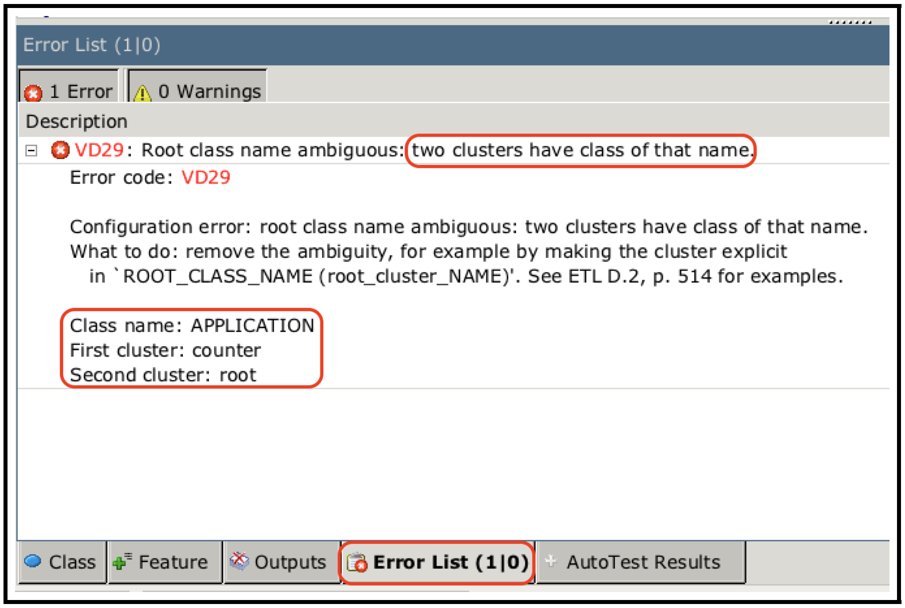
Name: root

Location: ./root

It is critical to understand that the `.` which we specify as the beginning of the location path designates the project location, i.e., where all bits and pieces of the current project are stored: `/Desktop/EECS3311_Labs/Lab_0/counter`. Consequently, when we specify that the `root` cluster has the location `./root`, we specify its path **relatively**: the `root` cluster corresponds to the physical directory `root` which resides right under the project location (i.e., `.`). **You should never ever specify an absolute path for your cluster location, because when you submit your projects for grading, that absolute path will just change and thus become invalid, meaning that your project will not compile anymore. When this happens, you will lose many marks for not being careful about this! PLEASE, BE CAUTIONS ABOUT THIS, ALWAYS SPECIFY RELATIVE PATHS FOR**

CLUSTERS.

- Proceed the same steps to create two other clusters `model` (with location `./model`) and `tests` (with location `.tests`).
- So now you should have four clusters: cluster `counter` (`./`), cluster `root` (`./root`), cluster `model` (`./model`), and cluster `tests` (`./tests`). If so, click on `Ok` on the Project Settings pop-up window and go compile your project. You should get a compilation error from the `Error List` panel:

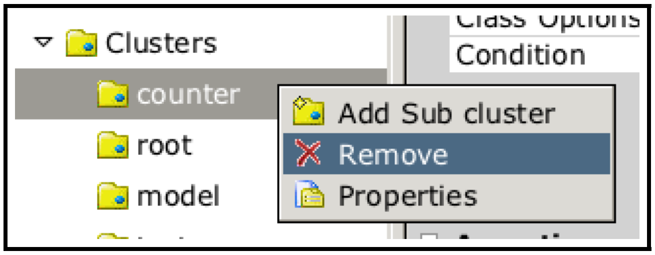


```
Error List (1|0)
1 Error   0 Warnings
Description
VD29: Root class name ambiguous: two clusters have class of that name.
    Error code: VD29

    Configuration error: root class name ambiguous: two clusters have class of that name.
    What to do: remove the ambiguity, for example by making the cluster explicit
      in `ROOT_CLASS_NAME (root_cluster_NAME)'. See ETL D.2, p. 514 for examples.

    Class name: APPLICATION
    First cluster: counter
    Second cluster: root
```

Let's try to understand why such an error (which occurs quite common!). In your file system, a file (e.g., `./root/application.e`) cannot exist in two different directories at the same time. However, recall that clusters in EStudio are only symbolic links that correspond the physical directories in your file system. Consequently, the directories that two clusters correspond to might overlap: for example, cluster `counter` has location `./` which completely overlaps with cluster `root` that has location `./root`. This means, even though the file `application.e` only exists in the directory `./root/application.e`, it is correct to say that it exists somewhere under `./` (the location of cluster `counter`) and it also exists somewhere under `./root/` (the location of cluster `root`). EStudio considers it as an error when an Eiffel source file exists under two different clusters. How do we fix this? Delete clusters so that the Eiffel source file `application.e` only exists in one cluster.

Let's remove the cluster `counter`, since its location (`./`) covers all subdirectories of the project location, so we will run into similar errors again. What we want to do is to make the structure of clusters in EStudio identical to the structure of directories in the file system.

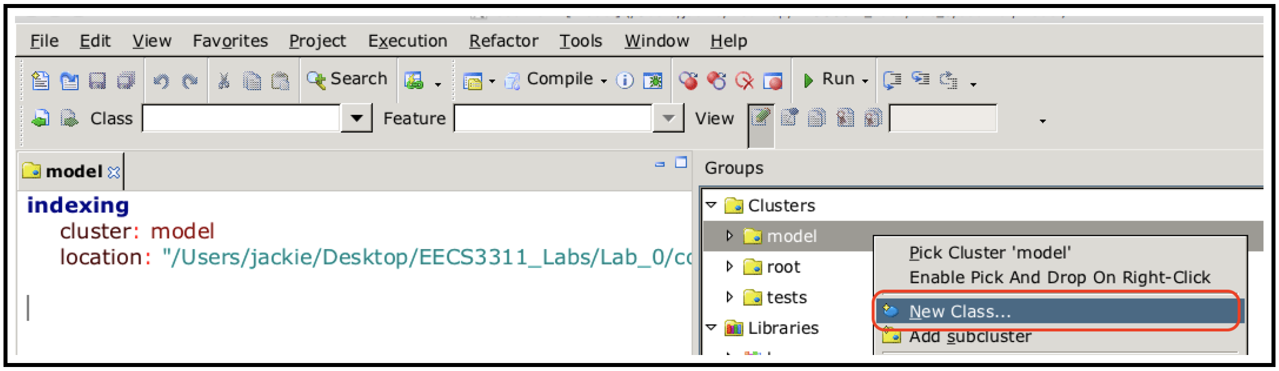
- Open the `Project Settings` window again. Under `Clusters`, right click on cluster `counter` and remove it.



Click on `Ok` and then recompile. Now everything should compile.

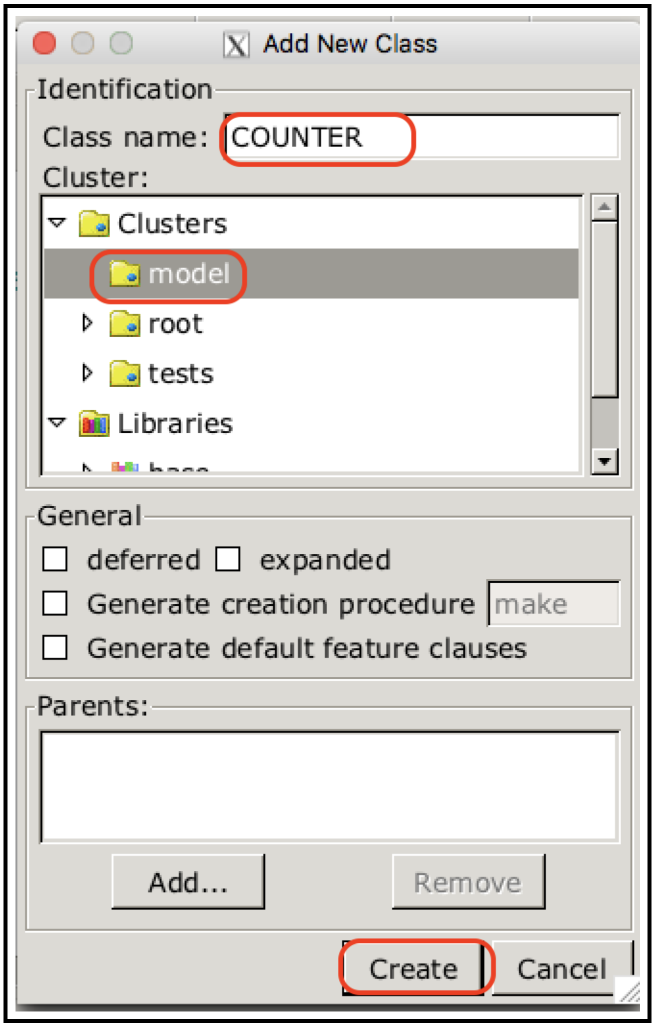
7 Creating a New Class

- In EStudio, on the right panel `Groups`, right click on cluster `model`, then select `New Class...`.



```
indexing
    cluster: model
    location: "/Users/jackie/Desktop/EECS3311_Labs/Lab_0/cc
```

- Enter the `Class name` as `COUNTER` and click on `Create`. In Eiffel convention, class names are all capitals, separated by underscores _ for compound words..



Class name: COUNTER

□ deferred □ expanded

□ Generate creation procedure make

□ Generate default feature clauses

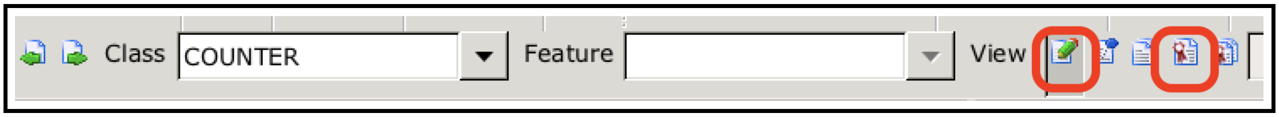
- Type in the following text for the `COUNTER` class (**don't be lazy, type!**):

```
note
  description: "A counter always has its value between 0 and 10."
  author: ""
  date: "$Date$"
  revision: "$Revision$"

class
  COUNTER
create -- We need to explicitly declare which feature is a constructor.
  make

feature -- Attribute: counter value
  value: INTEGER

feature -- Constructor
  make (v: INTEGER)
          -- Initialize counter with value ’v’.
    -- No require clause here means that there’s no precondition.
    -- Any input value ’v’ will be accepted and used to assign to ’value’.
    do
      value := v
    end

feature -- Commands (mutators in Java)
  increment_by(v: INTEGER)
      -- Increment the counter value by ’v’ if
      -- it causes its value to go above the max.
    require -- Precondtion: what’s assumed true by the supplier
      not_above_max: value + v <= 10
    do
      -- Implementation
      value := value + v
    ensure -- Postcondition: what’s expected true guaranteed by supplier
      value_incremented: value = old value + v
    end

  decrement_by (v: INTEGER)
      -- Decrement the counter value by ’v’ if
      -- it causes its value to go below the min.
    require
      not_below_min: value - v >= 0
    do
      value := value - v
    ensure
      value_decremented: value = old value - v
    end

invariant -- Class invarinat: what a legitimate counter means.
  counter_in_range:
    0 <= value and value <= 10
end
```

– Notice that line comments in Eiffel are preceded by `--`.

- Once you have typed the above Eiffel code, compile and make sure everything is ok.
- Study this code via the comments provided to you. Try to understand what's going on, especially how contracts (i.e., preconditions, postconditions, and class invariants) are specified.

  Hints: Under `Views`, switch between the `Basic text view` and `Contract view`:



What differences do you between these two views? Based on what we learned about Design by Contract, which view is **supplier**'s and which one is **client**'s?

8 Defining the APPLICATION Class

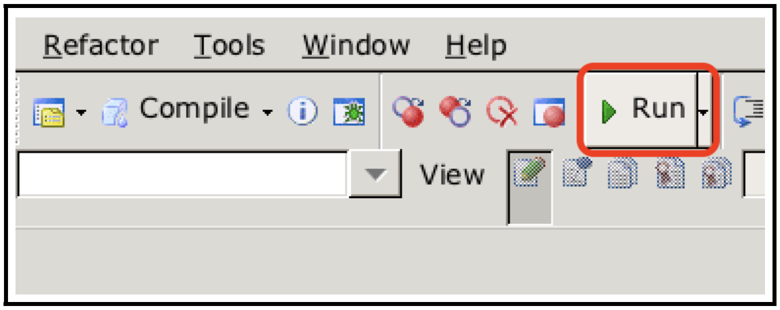
- Go to the `APPLICATION` class that we modified before, change its `make` feature so that it looks like:

```
make
    -- Run application.
  local -- local variables
    c: COUNTER
  do
    create {COUNTER} c.make (-10)
    print (c.value)
  end
```

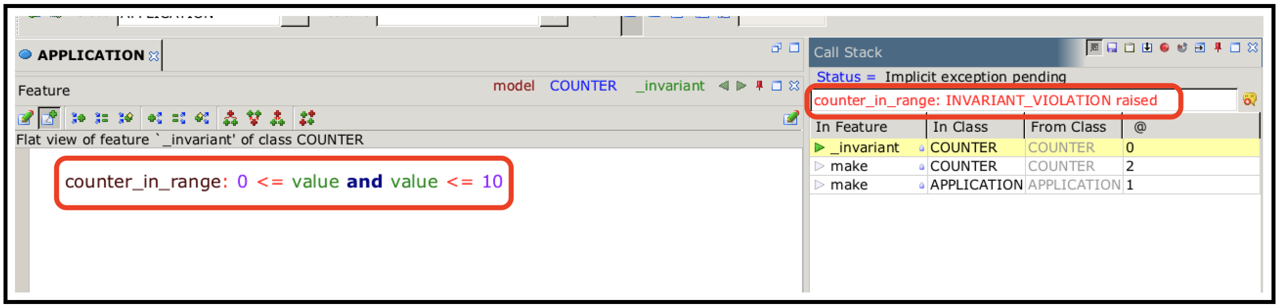
where

- In Eiffel, assignments are done using `:=`, whereas value comparisons are done using `=` (which corresponds more closely to math).
- The Eiffel syntax `c:  COUNTER` for declaring a variable corresponds to `COUNTER c` that you write in Java.
- The Eiffel syntax `create {COUNTER} c.make (-10)` for creating a new object of type `COUNTER` corresponds to `COUNTER c = new COUNTER(-10)` that you write in Java.

- Click on `Run` to execute the code.



- Then you should run into this contract violation (i.e., the class invariant of `COUNTER` fails to satisfy):



Flat view of feature `_invariant' of class COUNTER

```
counter_in_range: 0 <= value and value <= 10
```

Status = Implicit exception pending

counter_in_range: INVARIANT_VIOLATION raised

In Feature	In Class	From Class	@
_invariant	COUNTER	COUNTER	0
make	COUNTER	COUNTER	2
make	APPLICATION	APPLICATION	1

9 You Tasks

- Can you explain why a violation of class invariant occurs in the above application code?

  Hint: What does it mean to be a legitimate COUNTER instance and where is that explicitly defined as a **contract** (i.e., precondition, postcondition, or class invariant) in the COUNTER class.

- There are two ways to fix this class invariant (try both and verify that it does get rid of the contract violation):

  - From the supplier COUNTER side, does the precondition (specified using requir) have any missing cases of illegal values? If so, fix the current precondition.
  - From the client APPLICATION side, let them pass a legitimate value for initializing the counter.

- Modify either the APPLICATION class or the COUNTER, so that you can observe other kinds of violations:

  - **Precondition violation (caused by illegal inputs by client)**: When a feature call (or method call in Java) is passed with an input argument value that does not satisfy the Boolean condition under the **require** clause.
  - **Postcondition violation (caused by wrong implementation by supplier)**: When a feature call's input argument value satisfies its precondition, then after executing its implementation (i.e., what goes between **do** and **ensure**), the object state (i.e,. in this case the counter value) does not satisfy the Boolean condition under the **ensure** clause.

- We will continue from here in the lectures.

- By Lab 1 next week, make sure finish studying this tutorial series on DbC and TDD:

  https://www.youtube.com/playlist?list=PL5dxAmCmjv_6r5VfzCQ5bTznoDDgh__KS