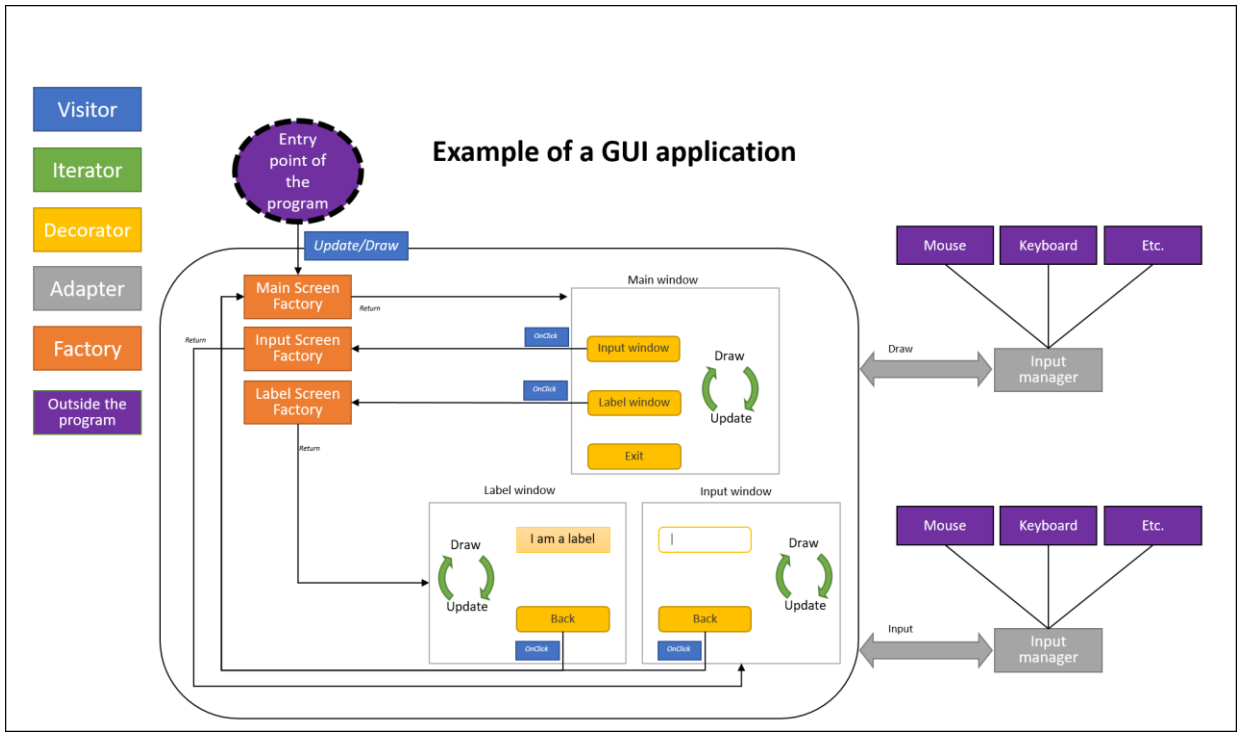
Assignment INFDEV02-4 + INFSEN01-2

In this assignment you will use several design patterns to support a primitive graphical user interface - or - implementation. Each week you will improve upon and refactor your existing implementation by applying a new pattern. Eventually, your application will contain a robust and reusable skeleton for a simple graphical user interface. This skeleton is documented in the diagram below.



Important to get the grade of the practicum you must finish this project before the deadline (see the end of this document). This means that all the patterns must work correctly and reflect the above diagram.

As a support we provide you with a series of iterations that are designed to ~gently~ introduce you to the project and to the design patterns involved. Every iteration comes with an associated incomplete solution that you can find on GitHub along with this document. These incomplete solutions are meant for you as guidelines to understand how to implement the project with actual code. For the final delivery, and in general for every iteration, you can wither use your own ~fixed~ version of these incomplete online solutions, or your own self-made custom solutions.

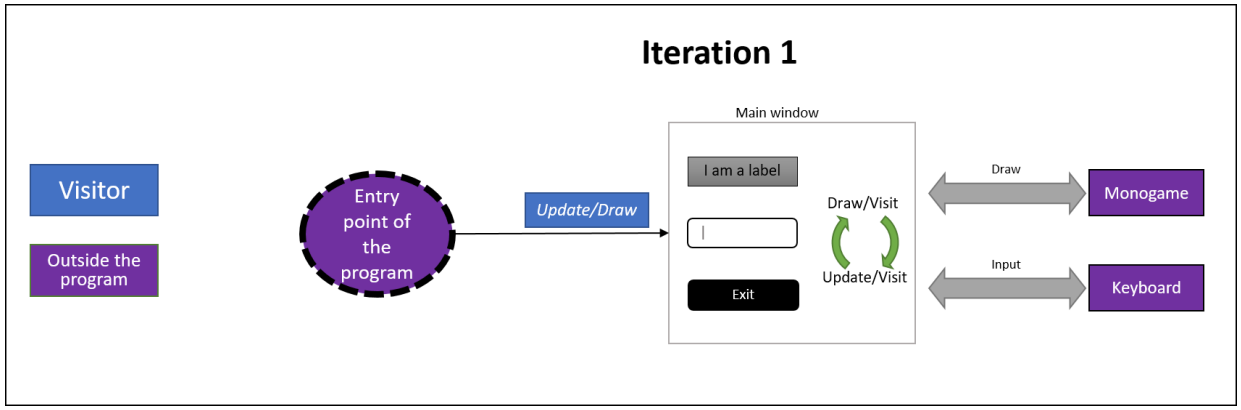
Iteration 1 - Visitor

Pattern discussed in lecture 1

With the visitor pattern you will specify a uniform interface for different polymorphic classes. This interface contains methods for each concrete class. These concrete classes in turn contain a 'visit' method that accepts the 'visitor' interface. The concrete classes then call the appropriate method on the visitor to trigger behavior specific to their concrete type.
For this week, your task is to implement a visitor pattern that draws a collection of GUI elements.

What to implement for this phase:

- A collection of concrete GUI element classes that implement a common interface
- A visitor interface for objects that implement the common interface
- A concrete visitor class that can visit each object implementing the common interface and draw the relevant item to the screen



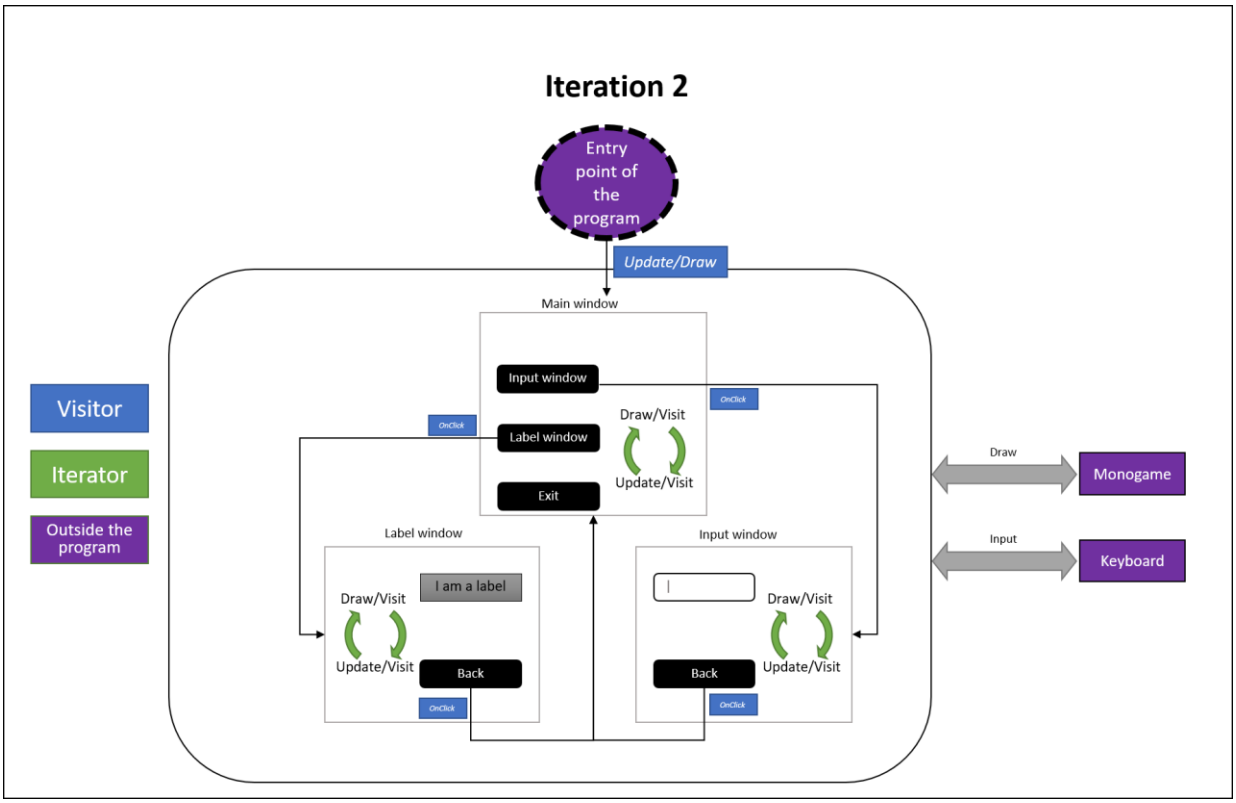
Iteration 2 - Iterator

Pattern discussed in lecture 2

This design pattern provides the foundations for iterable collections. Using the pattern, you will write your own - custom - iterable collection that contains all of the GUI elements in your application. In this version of the program, each GUI element should respond to input events.

What to implement for this phase:

- A custom iterator that can contain and provide all GUI elements. This iterator should basically replace your List or Array
- Using your custom iterator, visit each GUI element with your existing visitor to draw their respective representations
- At the beginning of each frame, check for mouse input and save the location of the cursor when the left mouse button is clicked
- Using your custom iterator, visit each GUI element with a new visitor that checks if the element was clicked by checking if the recorded location of the cursor intersects the element's boundaries



Iteration 3 - Adapter

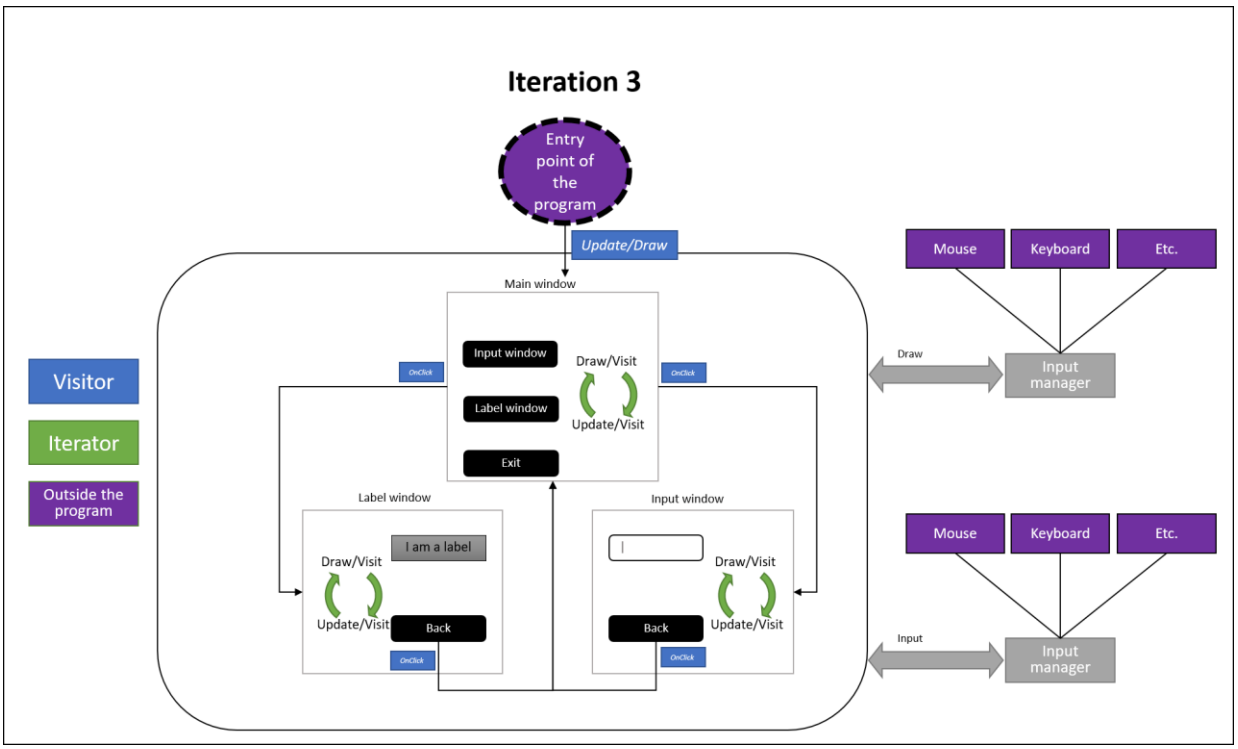
Pattern discussed in lecture 3

The adapter pattern involves writing a reliable interface to an existing piece of code. It may for instance be used to avoid invalid interaction with a poorly written class. When you do not have the freedom to modify code you must interact with, the adapter pattern is very useful.

In this phase you will use the adapter pattern to abstract away framework-specific (such as MonoGame or LibGDX) code. As a result, your code will be largely agnostic to whatever framework is used to perform such operations as rendering (drawing) and managing input.

What to implement for this phase:

- Use the adapter pattern to encapsulate any code calling the external framework (such as drawing functions)
- Refactor your code so that it provides only framework-agnostic data to the adapter. The adapter, in turn, will call the actual external framework's functions



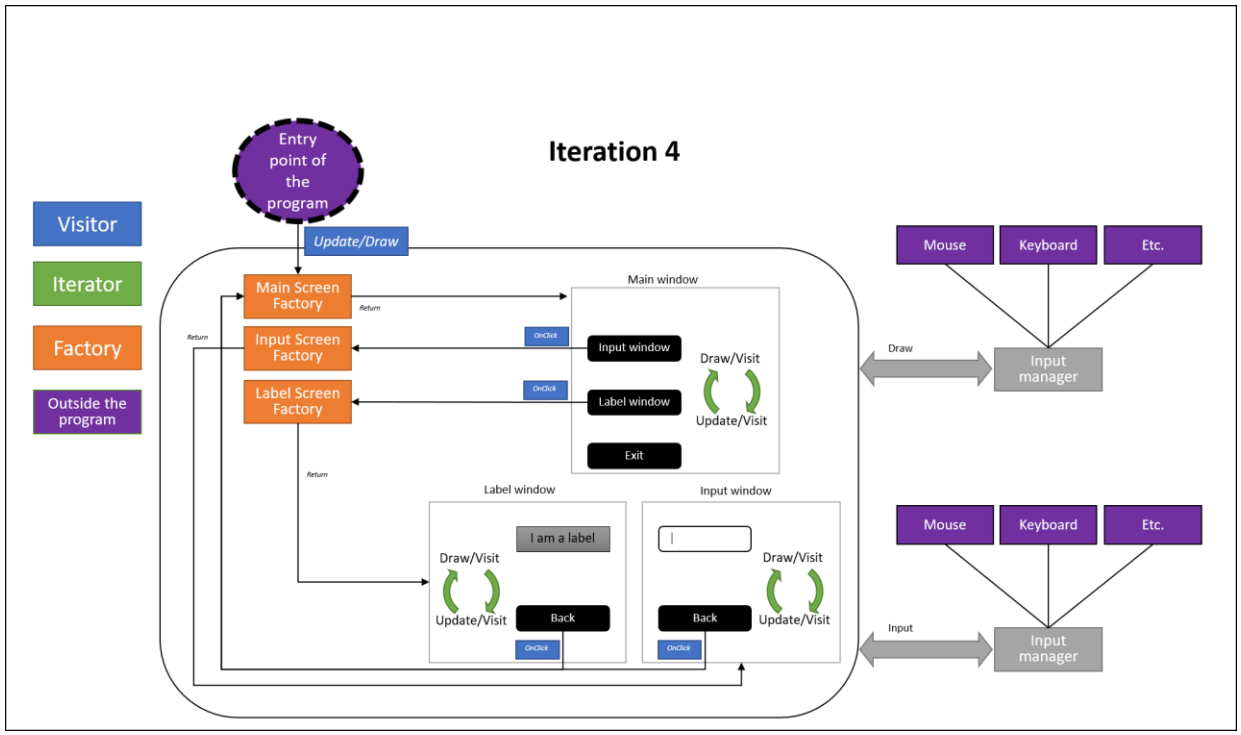
Phase 4 - Factory method

Pattern discussed in lecture 4 (A and B)

This design pattern provides a flexible way to create objects without specifying their concrete types. By delegating the instantiation of objects to the virtual factory method, the concrete types can be specified later.

What to implement for this phase:

- Create an abstract class with a virtual factory method for each view in your application. For example: main menu view, settings menu view, confirmation dialog view, etcetera
- Provide a realization of the abstract class that implements each virtual method. These virtual methods will decide which elements should be instantiated for the respective view
- Ensure that each view is populated with a few GUI elements



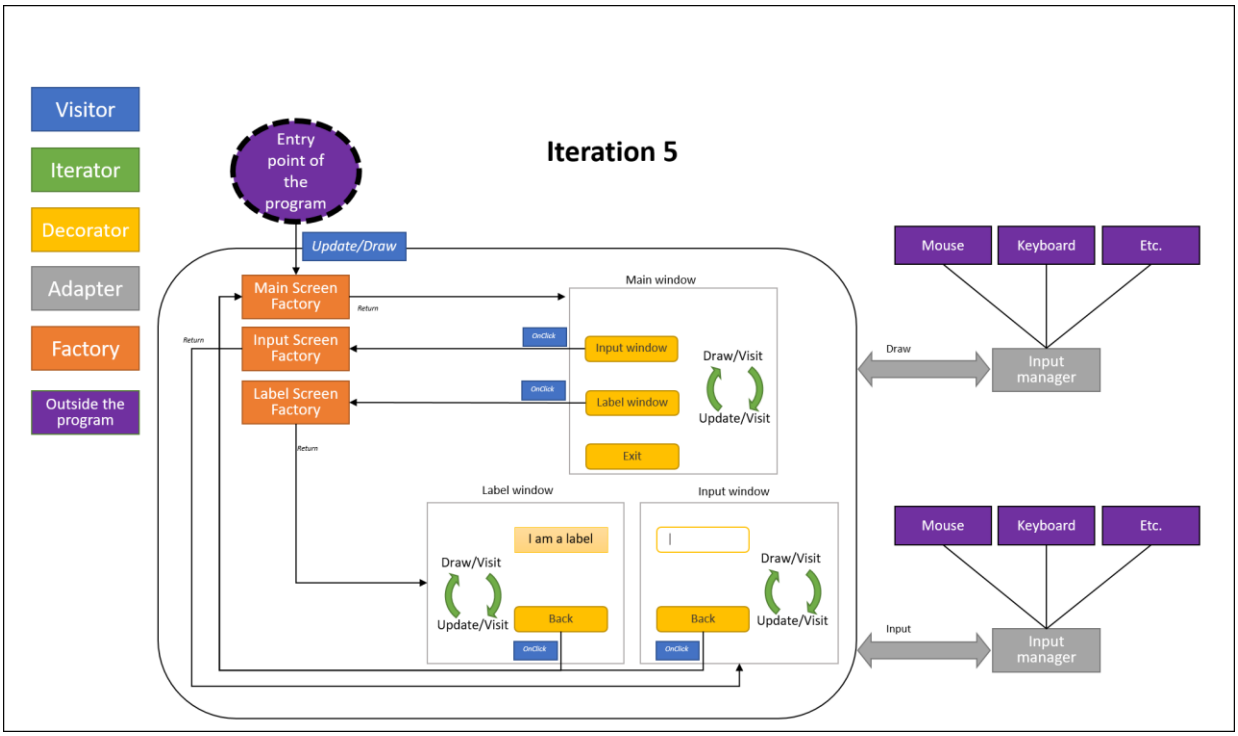
Phase 5 - Decorator

Pattern discussed in lecture 5

The decorator pattern enables you to enrich the functionality of objects in your application at runtime. Rather than increasing complexity by subclassing, the decorator pattern involves enclosing objects in 'decorations' that attach additional features to the containing object. Decorators are designed to be composable, meaning more than a single 'decoration' may be attached to a supported object.

What to implement for this phase:

- Apply the decorator pattern such that with composition of decorators, useful GUI elements may be created. For instance, composing a *Label* decorator with a *Clickable* decorator might result in a button.
- Refactor all your GUI elements such that they support the decorator pattern
- Actually implement the Clickable decorator so that your buttons are interactive



Additional info

Deadline:

• Week 7 OP4 (12 hours before the practicum check)

***Hand-in*:**

• on N@tschool -> INFDEV02-4 -> Assignment dev 4 - GUI app (for dev 4 students)
• on N@tschool -> INFSEN01-2 -> Assignment sen 2 - GUI app (for sen 2 students)

Remember:

the assignment is individual even though you can work in group. In general, whether you work in group, or alone, at the end you must be able to explain the code you delivered and justify the choices made.