ITI 1121. Introduction to Computing II

Graphical user interface : **Model-View-Controller**

by

Marcel Turcotte

Version April 4, 2020

Preamble

Preamble

Overview

Overview

Graphical user interface : Model-View-Controller

In software development, a **design pattern** is a specific arrangement of classes. It's a standard way to solve well-known problems. Programmers in the industry are **familiar** with these patterns. This week, we discover the design pattern **Model-View-Controller (MVC)** which is used in the development of graphical user interfaces.

General objective:

- This week, you will be able to design the graphical user interface of a simple application by applying the Model-View-Controller design pattern.

Preamble

Learning objectives

Learning objectives

- **Describe** in your own words the Model-View-Controller design pattern.
- **Use** the Model-View-Controller design pattern to produce the visual rendering of a graphical user interface.

Readings:

- `https://en.wikipedia.org/wiki/Model-view-controller`
- `http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf`

Preamble

Plan

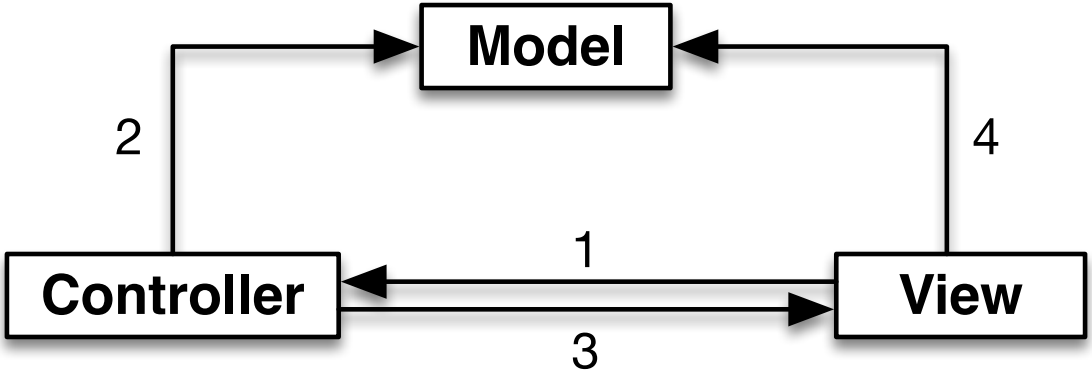
Plan

Theory

Design pattern

- In software development, a **design pattern** is a specific arrangement of classes.
- It's a **standard** way to solve well-known problems.
- Programmers in the industry are **familiar** with these patterns.

Model-View-Controller



- **MVC** separates the **data**, the **view** and the **logic** from the application.

Benefits

- Allows you to modify or adapt each part of the application **independently**;
- Promotes the implementation of **several views**;
- The **association between the model and the view is done dynamically** at runtime (not at compile time), which allows the view to be changed during execution.

Definitions

- **Model** – implementation, state: attributes and behaviors;
- **View** – the output interface, a representation of the model for the outside world;
- **Controller** – the input interface, routes user requests to update the model.

Model

- Contains the **application data** and methods for transforming the data;
- Possesses **a minimal knowledge of the graphical interface** (sometimes no knowledge at all);
- The **view** and the **model** are very **different**.

View

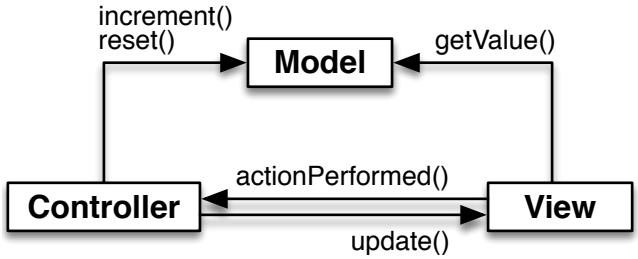
- A **representation** (graphical, textual, vocal, etc.) of the model for the **user**.

Controller

- A controller is an object that allows the user to **transform** the **data** or the **representation**;
- **Knows the model very well**.

Example

Counter



- A graphical interface for the class **Counter**.

Counter: Model

```
public class Counter {
    private int value;
    public Counter() {
        value = 0;
    }
    public void increment() {
        value++;
    }
    public int getValue() {
        return value;
    }
    public void reset() {
        value = 0;
    }
    public String toString() {
        return "Counter: {value="+value+"}";
    }
}
```

View

```
public interface View {
    void update();
}
```

- To facilitate the development of multiple views, we create the interface **View**.
- Our example will have two views: **GraphicalView** and **TextView**.

```java
public class TextView implements View {

    private Counter model;

    public TextView(Counter model) {
        this.model = model;
    }

    public void update() {
        System.out.println(model.toString());
    }

}
```

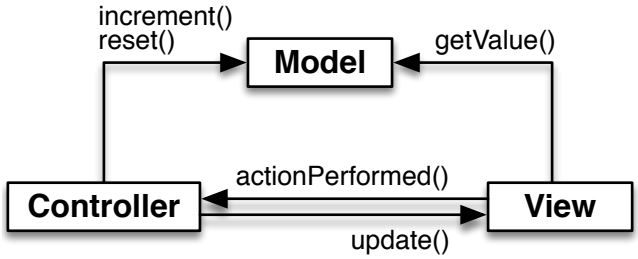
```
public class GraphicalView extends JFrame implements View {
    private JLabel input;
    private Counter model;
    public GraphicalView(Counter model, Controller controller) {
        setLayout(new GridLayout(1,3));
        this.model = model;
        JButton button;
        button = new JButton("Increment");
        button.addActionListener(controller);
        add(button);
        JButton reset;
        reset = new JButton("Reset");
        reset.addActionListener(controller);
        add(reset);
        input = new JLabel();
        add(input);
    }
    public void update() {
        input.setText(Integer.toString(model.getValue()));
    }
}
```

```
public class Controller implements ActionListener  {

    private Counter model;

    private View[] views;
    private int numberOfViews;

    public Controller() {

        views = new View[2];
        numberOfViews = 0;

        model = new Counter();

        register(new GraphicalView(model, this));
        register(new TextView(model));

        update();

    }
```

```
    private void register(View view) {
        views[numberOfViews] = view;
        numberOfViews++;
    }

    private void update() {
        for (int i=0; i<numberOfViews; i++) {
            views[i].update();
        }
    }
```

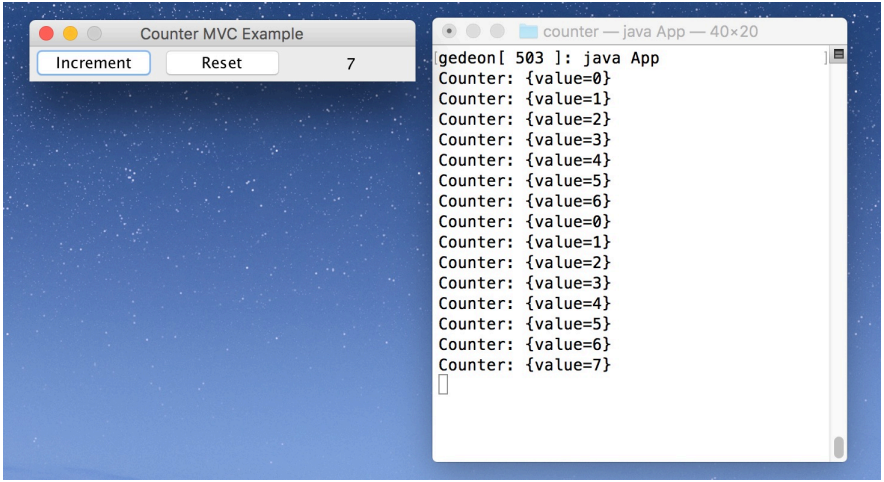
```
    public void actionPerformed(ActionEvent e) {

        if (e.getActionCommand().equals("Increment")) {
            model.increment();
        } else {
            model.reset();
        }

        update();
    }
}
```

Application Counter



```
public class App {

    public static void main(String[] args) {

        Controller controller;
        controller = new Controller();

    }

}
```

Counter



- The application **Counter** and its two views: textual and graphical.

Prologue

Summary

- The **MVC** clearly separates the **data**, the **views**, and the **logic** of an application.

Excercises

Implement each of the following applications using the Model-View-Controller (MVC) design pattern.

- A game of **tic-tac-toe**
- A game of **battle ship**
- A game of **memory**
- Game 2048

Next module

-

References I

E. B. Koffman and Wolfgang P. A. T.
Data Structures: Abstraction and Design Using Java.
John Wiley & Sons, 3e edition, 2016.

Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

School of Electrical Engineering and **Computer Science** (EE**CS**)

University of Ottawa