Design Patterns

Gerwin van Dijken (gerwin.vandijken@inholland.nl)

Programma periode 1.4

01 (wk-15)	abstracte classes en interfaces
02 (wk-16)	Template Method pattern / Observer pattern
03 (wk-17)	MVC pattern
04 (wk-18)	*geen lessen (meivakantie)*
05 (wk-19)	Strategy pattern / Adapter pattern
06 (wk-20)	Singleton pattern / State pattern
07 (wk-21)	Factory patterns
08 (wk-22)	herhaling / proeftentamen

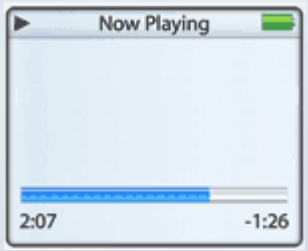
---

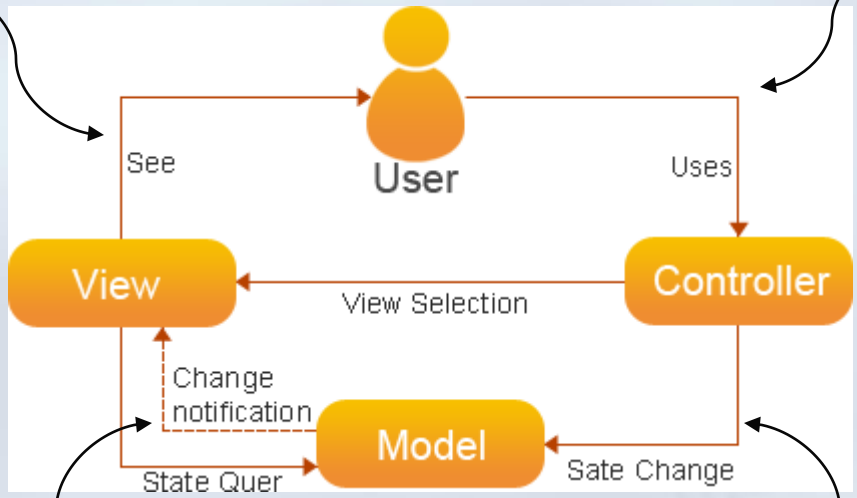
09 (wk-23)	tentamen (praktijk)
10 (wk-24)	*hertentamens (vakken periode 1.3)*
11 (wk-25)	*hertentamens (vakken periode 1.4)*

MVC – Model View Controller

Gebruiker ziet via de View(s) de nieuwe status van het model.

Gebruiker gebruikt de Controller om bv een nieuw nummer te starten.





Model laat View weten dat er een nieuw nummer is begonnen.

Controller geeft aan Model door dat volgend nummer gestart moet worden.

```
class MP3Player
{
    private List<Song> playList;
    private Song currentSong;
    // ...

    public void Play() { }
    public void Stop() { }
    public void Next() { }
    public void SetVolume(int level) { }
}
```

Een voorbeeld applicatie

- Een MP3-player met een playlist met functies zoals Play / Stop / Next (model)
- Een controller om de MP3-player te bedienen (play/stop, next, volume)
- Een display om het huidige nummer te tonen, en een aparte display om het volume te tonen (views)

Model: MP3-player

- We gebruiken een interface voor het model
- De controller en view kennen deze interface

```
public interface IMP3Player
{
    void Play();
    void Stop();
    void Next();
    void SetVolume(int volumeLevel);

    Song CurrentSong { get; }
    bool IsPlaying { get; }
    int VolumeLevel { get; }

    void AddObserver(ISongObserver observer);
    void RemoveObserver(ISongObserver observer);
    void AddObserver(IVolumeObserver observer);
    void RemoveObserver(IVolumeObserver observer);
}
```

Via deze methoden kan de MP3-player bediend worden.

Er zitten een aantal read-only properties in de interface.

Deze methoden zijn nodig voor het aan-/afmelden van observers. We gebruiken hier 2 soorten observers.

Model: 'the real thing' (1/3)

```
class MP3Player : IMP3Player
{
    private List<Song> playList;
    private Song currentSong;
    private bool isPlaying;
    private int volumeLevel;

    private List<ISongObserver> songObservers;
    private List<IVolumeObserver> volumeObservers;

    public Song CurrentSong { get { return currentSong; } }
    public bool IsPlaying { get { return isPlaying; } }
    public int VolumeLevel { get { return volumeLevel; } }

    // ...
```

Members van de MP3-player.

2 aparte lijsten voor de observers.

Properties van de MP3-player.

IMP3Player is het 'contract' waaraan class MP3Player zich moet houden...

Model: 'the real thing' (2/3)

In de constructor maken we de 2 observer-lijsten aan.

Zodra er iets met het huidige nummer gebeurd (play, stop of next), worden alle SongObservers op de hoogte gebracht.

Zodra het volume wijzigt, worden alle VolumeObservers op de hoogte gebracht.

```
public MP3Player() {
    // ...

    // make observer lists
    songObservers = new List<ISongObserver>();
    volumeObservers = new List<IVolumeObserver>();
}

public void Play() {
    // ...
    NotifySongObservers();
}

public void Stop() {
    // ...
    NotifySongObservers();
}

public void Next()
{
    // ...
    NotifySongObservers();
}

public void SetVolume(int volumeLevel)
{
    // ...
    NotifyVolumeObservers();
}
```

Model: 'the real thing' (3/3)

Public methoden voor SongObservers (Add & Remove).

Public methoden voor VolumeObservers (Add & Remove).

Private methoden voor Notify_X_Observers.

```
    // ...

    public void AddObserver(ISongObserver observer) {
        songObservers.Add(observer);
    }

    public void RemoveObserver(ISongObserver observer) {
        songObservers.Remove(observer);
    }

    public void AddObserver(IVolumeObserver observer) {
        volumeObservers.Add(observer);
    }

    public void RemoveObserver(IVolumeObserver observer) {
        volumeObservers.Remove(observer);
    }

    private void NotifySongObservers() {
        foreach (ISongObserver observer in songObservers)
            observer.Update(this.currentSong);
    }

    private void NotifyVolumeObservers() {
        foreach (IVolumeObserver observer in volumeObservers)
            observer.Update(this.volumeLevel);
    }
}
```

Controller

- Ook voor de Controller gebruiken we een interface (contract)
- Met de Controller wordt de Model gemanipuleerd

```
public interface IMP3Controller
{
    void Play();
    void Stop();
    void Next();
    void VolumeUp();
    void VolumeDown();
}
```

Controller

MP3Controller implementeert IMP3Controller.

De IMP3Player wordt via de constructor meegegeven.

Alle akties worden vanuit MP3-Display (Cntrl) aange-roepen.

Sommige akties worden 1-op-1 doorgegeven aan de player.

Andere akties moeten enigzins aangepast doorgegeven worden.

```
public class MP3Controller : IMP3Controller
{
    private IMP3Player player;

    public MP3Controller(IMP3Player player) {
        this.player = player;
    }

    public void Play() {
        player.Play();
    }

    public void Stop() {
        player.Stop();
    }

    public void Next() {
        player.Next();
    }

    public void VolumeUp() {
        int volumeLevel = player.VolumeLevel;
        player.SetVolume(++volumeLevel);
    }

    public void VolumeDown() {
        int volumeLevel = player.VolumeLevel;
        player.SetVolume(--volumeLevel);
    }
}
```

View: Observer interfaces

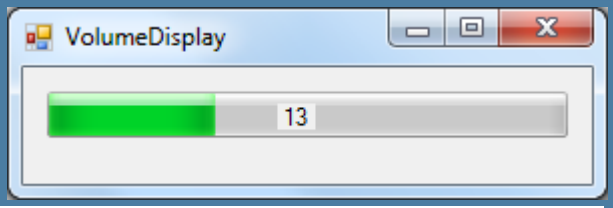
- ISongObserver: nummer wijziging
- IVolumeObserver: volume wijziging

```
public interface ISongObserver
{
    void Update(Song currentSong);
}
```

```
public interface IVolumeObserver
{
    void Update(int currentVolume);
}
```

- Zou één Observer interface ook kunnen volstaan?

View: VolumeDisplay



VolumeDisplay implementeert interface IVolumeObserver.

Deze display krijgt de IMP3Player mee via de constructor.

Deze display meldt zich direct aan als VolumeObserver...

...zodat deze Update-methode wordt aangeroepen door het Model (=IMP3Player), zodra het volume wijzigt.

```
public partial class VolumeDisplay : Form, IVolumeObserver
{
    IMP3Player player;

    public VolumeDisplay(IMP3Player player)
    {
        InitializeComponent();

        this.player = player;
        this.player.AddObserver(this);
    }

    private void VolumeDisplay_Load(object sender, EventArgs e)
    {
        // initialize progress bar
        progressBar1.Minimum = 0;
        progressBar1.Maximum = 40;
        progressBar1.Value = player.VolumeLevel;
        lblVolumeLevel.Text = player.VolumeLevel.ToString();
    }

    public void Update(int currentVolume)
    {
        progressBar1.Value = currentVolume;
        lblVolumeLevel.Text = currentVolume.ToString();
    }
}
```

View: MP3Display

MP3Display implementeert interface ISongObserver en IVolumeObserver.

```
public partial class MP3Display : Form, ISongObserver, IVolumeObserver
{
    private IMP3Player player;
    private IMP3Controller controller;

    public MP3Display(IMP3Player player, IMP3Controller controller) {
        InitializeComponent();

        this.player = player;
        this.controller = controller;

        this.player.AddObserver((ISongObserver)this);
        this.player.AddObserver((IVolumeObserver)this);
    }

    public void Update(Song currentSong) {
        // update song information
        if (currentSong == null)
            lblCurrentSong.Text = "Not playing...";
        else
            lblCurrentSong.Text =
                String.Format("{0} ({1})", currentSong.Title, currentSong.Artist);
    }

    public void Update(int currentVolume)
    {
        lblVolume.Text = currentVolume.ToString();
    }
```

Deze display krijgt de IMP3Player en IMP3Controller mee via de constructor.

Display meldt zich direct aan als SongObserver en als VolumeObserver...

... zodat deze 2 Update methoden worden aangeroepen.

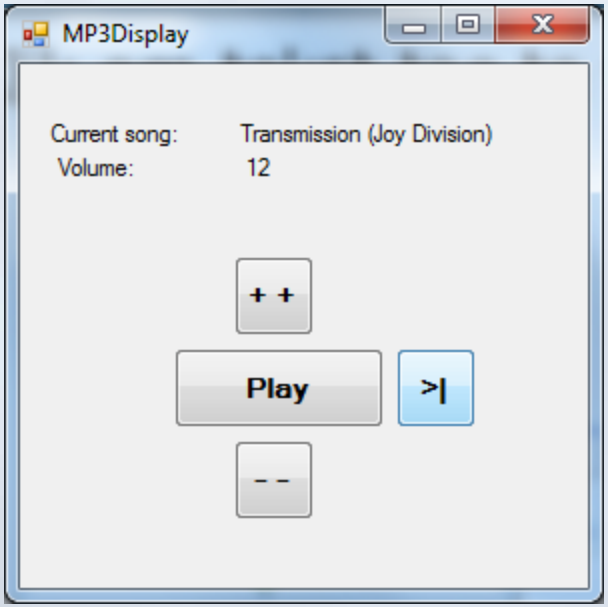
MP3Display

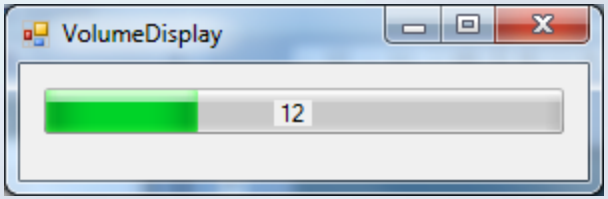
Alle user-input wordt doorgesluist naar de controller.

Ook de volume regeling wordt doorgesluist naar de controller.

```
// ...

private void btnStopPlay_Click(object sender, EventArgs e)
{
    if (player.IsPlaying)
        // pass 'stop' action on to controller
        controller.Stop();
    else
        // pass 'play' action on to controller
        controller.Play();
}

private void btnNext_Click(object sender, EventArgs e)
{
    // pass 'next' action on to controller
    controller.Next();
}

private void btnVolumeUp_Click(object sender, EventArgs e)
{
    // pass 'volume up' action on to controller
    controller.VolumeUp();
}

private void btnVolumeDown_Click(object sender, EventArgs e)
{
    // pass 'volume down' action on to controller
    controller.VolumeDown();
}
```

Displays


MP3Display
Current song:
Transmission (Joy Division)
Volume:
12
+ +
Play
>|
- -


VolumeDisplay
12

Samenvattend

- Met de MVC pattern scheiden we een aantal zaken: de data/het model, de presentatie van het model, de verwerking van userinput / model aanpassingen
- Het model wordt alleen gemanipuleerd door de controller
- Events vanuit display worden doorgesluisd/gedelegeerd naar de controller

Opdrachten

- Zie Moodle: ‘Week 3 opdrachten’