

12 The MVC model

Main concepts to be covered

- Design patterns
- The **Observer** design pattern
- The **Model View Controller** architecture

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 2

Using design patterns

- Inter-class relationships are important, and can be complex.
- Some relationship recur in different applications.
- Design patterns help clarify relationships, and promote reuse.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 3

Pattern structure

- A pattern name.
- The problem addressed by it.
- How it provides a solution:
 - Structures, participants, collaborations.
- Its consequences.
 - Results, trade-offs.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 4

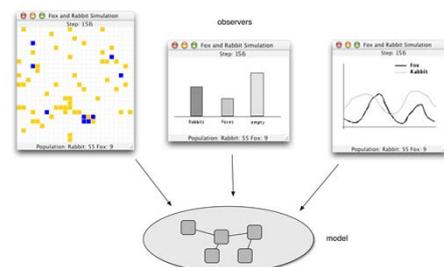
Design pattern: Observer

- Supports separation of internal model from a view of that model.
- Observer defines a one-to-many relationship between objects
 - *publisher - subscriber*
- The object-observed notifies all Observers of any state change.
- Example `SimulatorView` in the *foxes-and-rabbits project*.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 5

Observers



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 6

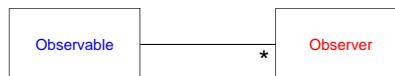
Main classes of interest

- **class java.util.Observable**
 - Subclasses inherit basic functionality for reporting state changes to observing objects.
 - Independent of the observer's logic
- **interface java.util.Observer**
 - Subclasses implement update functionality.
 - Many objects can connect to the same observable object.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 7

Class relationships

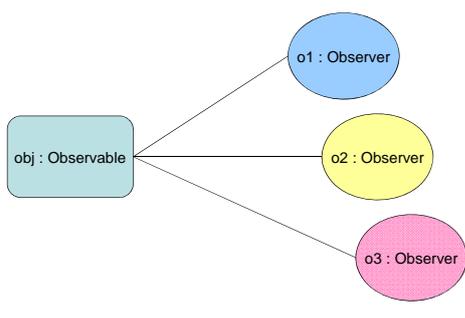


- No particular owner-owned relationship
 - Observers do not own the observed objects.
 - Observed objects are *unaware* of observers.
 - The relation is *navigable* in both directions
 - *Observers know what they observe.*
 - *Observables must be able to update observers (weak dependency).*

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 8

Typical configuration



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 9

class Observable

```
public class Observable {
    - Add observer o to the set of observers for this object
    public void addObserver(Observer o)

    - Mark this object as changed
    public void setChanged()

    - If this object has changed, then notify all of it's observers
    public void notifyObservers()
}
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 10

interface Observer

```
public interface Observer {

    An observable object calls it's inherited notifyObservers method
    to have all the object's observers notified of a state change.
    notifyObservers then calls update for each observer.

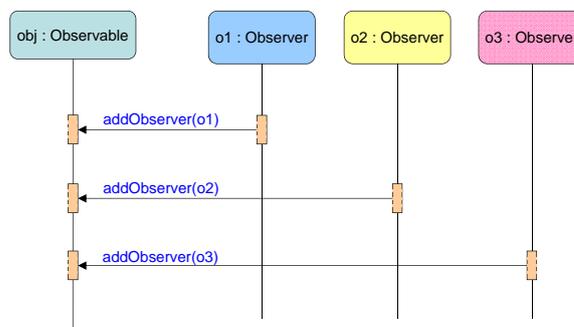
    Parameters:
    o - the observable object who initiated the call.
    arg - the argument that was passed to the
    notifyObservers method by the observable object.
    notifyObservers forwards this argument to update.

    void update(Observable o, Object arg);
}
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 11

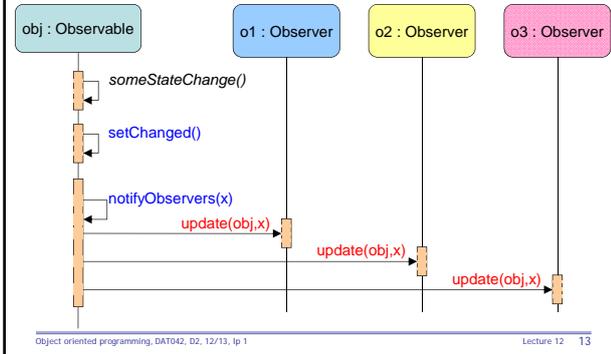
Observer registration



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 12

An update scenario



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 13

Typical Observable class

```

public class Obsrvbl extends Observable {
    private SomeType x;

    public void someMutator() {
        ...
        x = ...; // x has changed, inform observers
        setChanged();
        notifyObservers(x.clone());
        ...
    }
}
    
```

Pass some information to the observers.
Maybe a copy of x, or something else.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 14

Typical Observer class

```

public class Obsrvr implements Observer {
    ...
    public void update(Observable o, Object arg) {
        if ( o instanceof Obsrvbl &&
            arg instanceof SomeType) {
            SomeType x = (SomeType) arg;
            // take some appropriate action
            // based on the value of x
        } else
            ...
    }
}
    
```

Several objects of different types may be observed by the same observer. Moreover, each observed object may, depending on the situation, pass arguments of different types to update. Hence a case analysis may be necessary.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 15

Observer registration

```

Observable obj = new Obsrvbl();

Observer o1 = new Obsrvr();
Observer o2 = new Obsrvr();
Observer o3 = new Obsrvr();

obj.addObserver(o1);
obj.addObserver(o2);
obj.addObserver(o3);
    
```

Observer registration

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 16

Alternative observer registration

```

public class Obsrvr implements Observer {

    public Obsrvr(Observable x) {
        ...
        x.addObserver(this);
        ...
    }

    public void update(Observable o, Object arg) {
        ...
    }
}
    
```

Observer registration

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 17

Alternative observer registration

```

Observable obj = new Obsrvbl();

Observer o1 = new Obsrvr(obj);
Observer o2 = new Obsrvr(obj);
Observer o3 = new Obsrvr(obj);
    
```

Observer registration

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 12 18

The MVC architecture

- *Reenskaug 1979 (Smalltalk-80)*
- **Model** (content)
- **View** (appearance)
- **Controller** (user actions)

Object oriented programming, DAT042, D2, 12/13, Ip 1

Lecture 12 19

Model

- Model classes take care of data storing and processing
 - *business logic*
 - *domain logic*
 - *the "database"*

Object oriented programming, DAT042, D2, 12/13, Ip 1

Lecture 12 20

View

- View classes take care of visual aspects
 - *Visualization*
 - *User interface*
 - *"Model rendering"*
 - *A model can have many views*

Object oriented programming, DAT042, D2, 12/13, Ip 1

Lecture 12 21

Controller

- Controller classes take care of the control flow between model and view
 - *User actions*
 - *Event handling*
 - *Control flow*
 - *Communication*

Object oriented programming, DAT042, D2, 12/13, Ip 1

Lecture 12 22

Model (2)

- Model objects are
 - observable
 - *unaware* of controller and view part
- The model is *decoupled* from the view

Object oriented programming, DAT042, D2, 12/13, Ip 1

Lecture 12 23

View (2)

- View objects are
 - observers of model objects
 - weakly dependent on model and controller

Object oriented programming, DAT042, D2, 12/13, Ip 1

Lecture 12 24

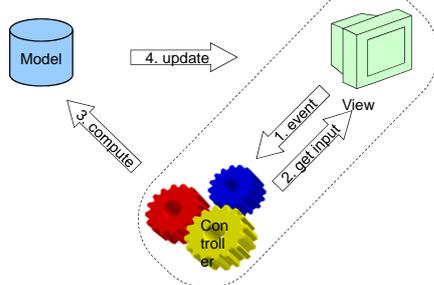
Controller (2)

- Controller objects
 - update the model with information obtained from the view
 - manage over all control flow, timing, etc
- Example: control flow in a game

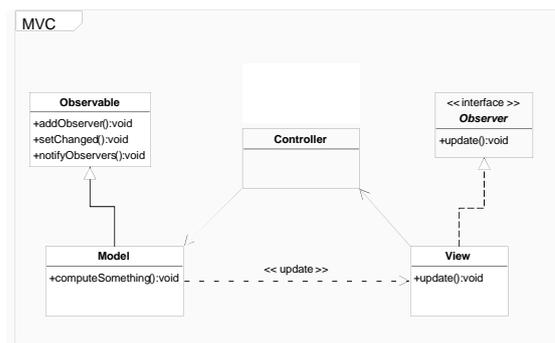
Variations

- Variations of the MVC pattern are possible.
- More or less coupling between model, view and controller:
 - View observes model directly.
 - or: Controller mediates all communication between model and view.

MVC architecture



MVC class diagram



Consequences

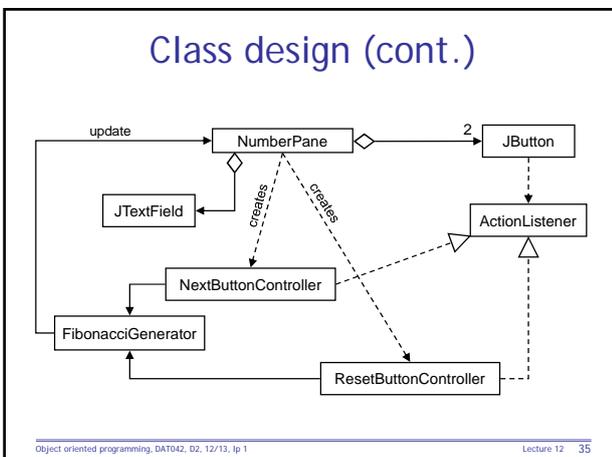
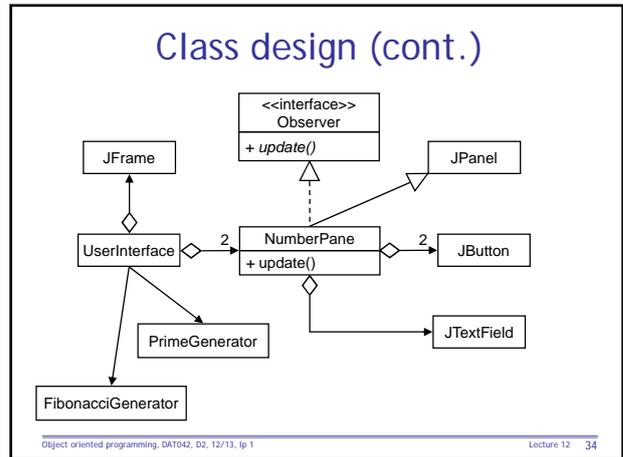
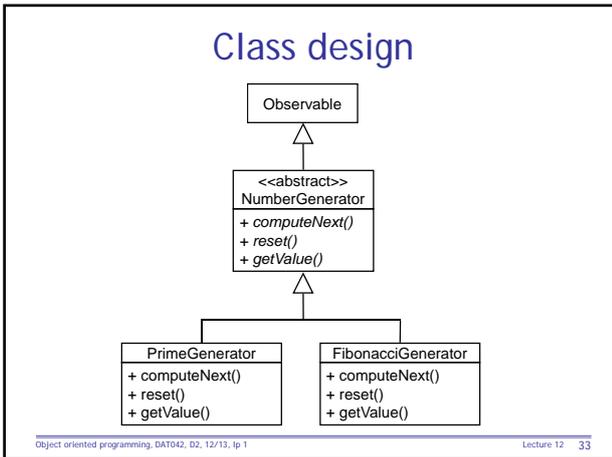
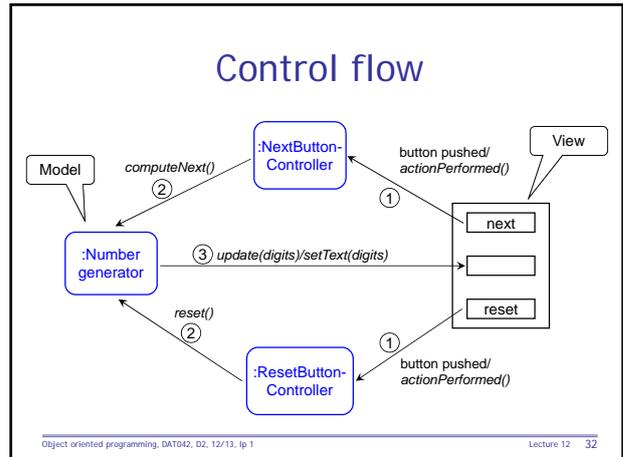
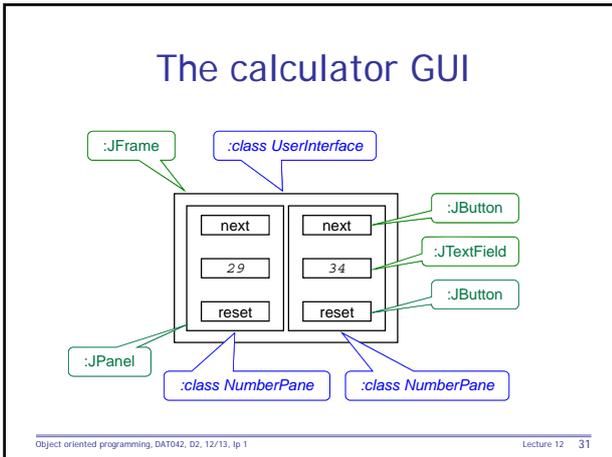
- + Model is completely independent of view.
- View is more or less dependent of model
 - the view must often have some *domain knowledge*. Eg. Syntax checking in forms.
- Controllers are dependent of model (and sometimes of view).

Example: Number series calculator

- A (very) simple calculator for exploring the prime number and Fibonacci number series.



- Program design based on the MVC pattern.
- Explore the *mvc1* project!



- ### Review
- The degree of dependency between components is called *coupling*.
 - Aim for less coupling!
 - The *observer* design pattern decreases coupling.
 - The MVC architectural pattern decouples the business logic from GUI issues
 - *thus easy to modify or replace GUI!*
- Object oriented programming, DAT042, D2, 12/13, lp 1 Lecture 12 36