The Observer/Observable Design Pattern

A simple application

- A boring application: nothing happens when you press the button

```
import javax.swing.JButton;
import javax.swing.JFrame;

public class SimpleApp {

    public static void main(String[] args) {
        JFrame frame=new JFrame("Hello");
        frame.setSize(400,200);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        JButton button=new JButton("Press me!");
        frame.add(button);

        frame.setVisible(true);
    }
}
```

Create a window

Create a button and add it to the window

Make the window visible

Reacting to button press

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class ButtonActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null,"Thank you!");
    }
}

public class AppWithActionListener {
    public static void main(String[] args) {
        JFrame frame=new JFrame("Hello");
        frame.setSize(400,200);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        JButton button=new JButton("Press me!");
        button.addActionListener(new ButtonActionListener());
        frame.add(button);

        frame.setVisible(true);
    }
}
```

Show a message box when the button is pressed

What‘s happening here?

How an ActionListener works

- ActionListener is an interface with one method. In our application, we have implemented that interface:

```
class ButtonActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null,"Thank you!");
    }
}
```

- We add our ButtonActionListener to the button:

```
button.addActionListener(new ButtonActionListener());
```

- What does that mean? We are telling the button:

"Dear button, please call the actionPerformed method of my action listener object when something interesting happens to you."

("Something interesting" = "Somebody clicked on you")

Observer and Observables

- In our previous example, the ActionListener interface was used to observe actions of the button

  ActionListener = *observing* actions

  Button click = *observable* action

- We can generalize this idea to any kind of things we want to observe
- Let's try it with this simple class. We would like to observe the value of the account.

```
public class Account {
    private int value ;

    public void deposit(int d) {
        value+=d;
    }

}
```

An observable account

- Here is the observer interface. We want that the method accountHasChanged is called when the account changes its value.

```
public interface AccountObserver {
    public void accountHasChanged(int newValue);
}
```

- Here is an observable version of the Account class:

```
public class ObservableAccount {
    private int value ;
    private AccountObserver observer;

    public void deposit(int d) {
        value+=d;
        if(observer!=null) {
            observer.accountHasChanged(value);
        }
    }

    public void setObserver(AccountObserver o) {
        observer=o;
    }
}
```

We have to check first whether there is an observer.

Everytime a deposit is made, we tell the observer

How to use it

```
public class MyObserver implements AccountObserver {
    @Override
    public void accountHasChanged(int newValue) {
        System.out.println("Account has changed. New value: "+newValue);
    }
}

public class MyMain {

    public static void main(String[] args) {
        ObservableAccount account=new ObservableAccount();
        MyObserver observer=new MyObserver();
        account.setObserver(observer);

        account.deposit(100);
        account.deposit(50);
    }
}
```

Observer/Observable: Summary

- The Observer/Observable design pattern allows you to separate objects containing data from the code reacting to changes in the data
  - The ObservableAccount class does not need to know what happens when the account value changes. We can change the code in the observer without changing the account implmentation.
- Disadvantage: The program becomes harder to understand
  - A simple call `account.deposit(100)` can have a lot of effects in the observer
- Note: Often, it would be useful to have multiple observers for one observable. So, instead of

```
public void setObserver(AccountObserver o)
```

we would like to have

```
public void addObserver(AccountObserver o)
```

See the exercise on inginious!