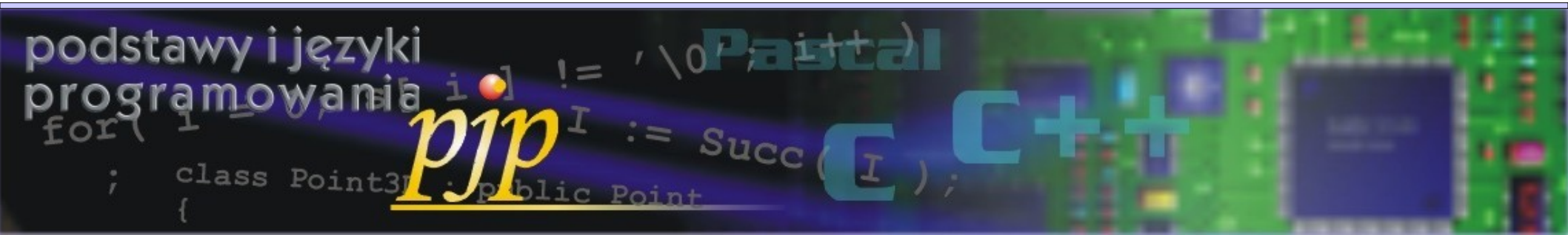
Projektowanie i programowanie obiektowe

Roman Simiński
roman.siminski@us.edu.pl
roman@siminskionline.pl
programowanie.siminskionline.pl

Wzorce projektowe

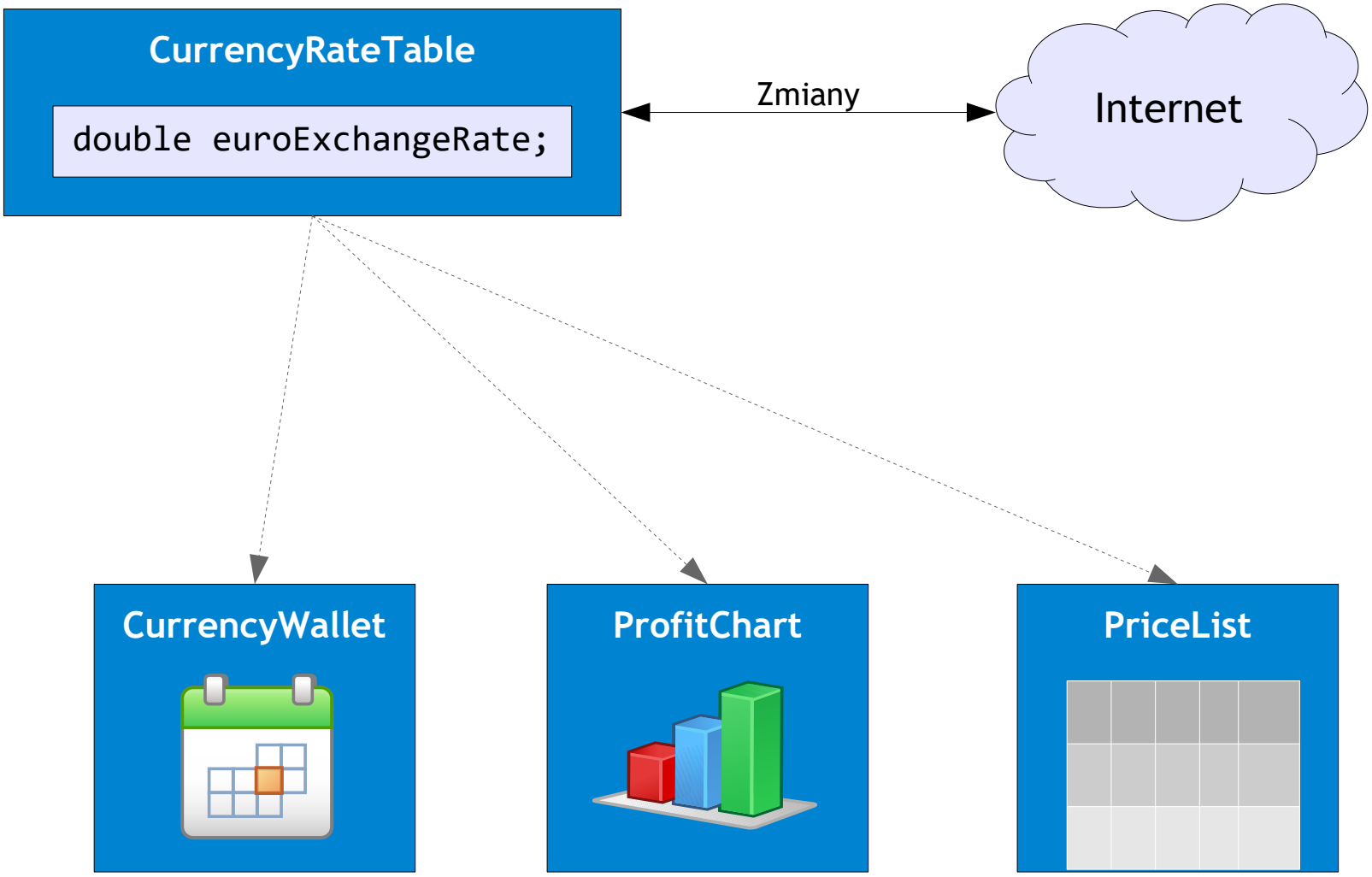
Wybrane wzorce operacyjne: Obserwator



Observator
Observer Pattern

Obserwator - koncepcja

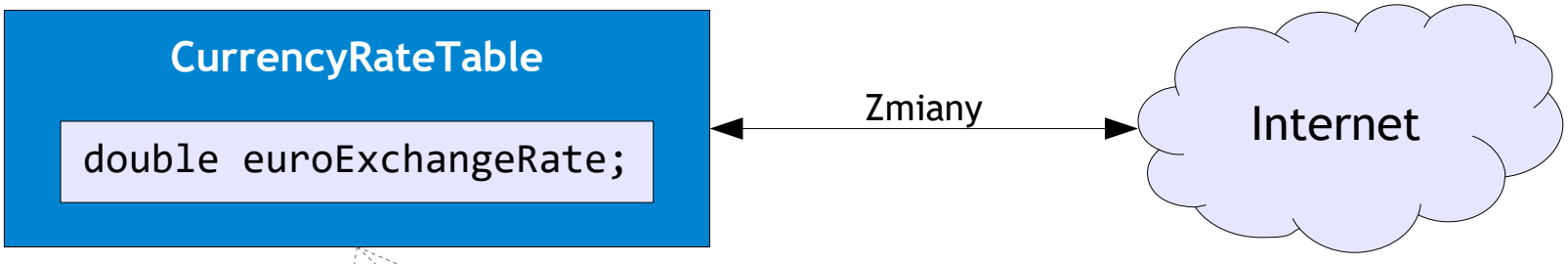
Obiekt ***podmiotu*** (***subject***) zawierający informację, której zmiana powinna być rozgłoszona



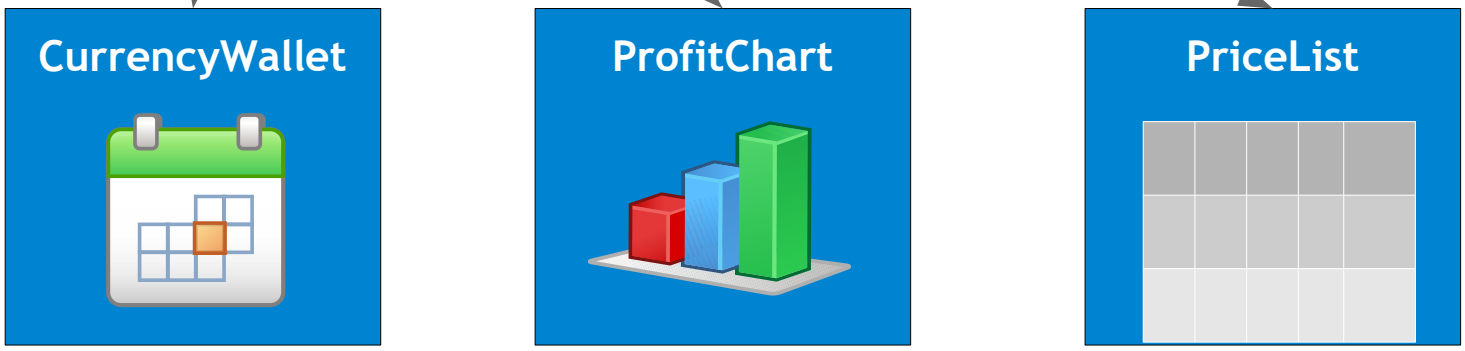
Obiekty, dla których zmiana stanu podmiotu jest istotna, powinny zostać o niej poinformowane

Obserwator - koncepcja

Obiekt ***podmiotu*** (***subject***) zawierający informację, której zmiana powinna być rozgłoszona



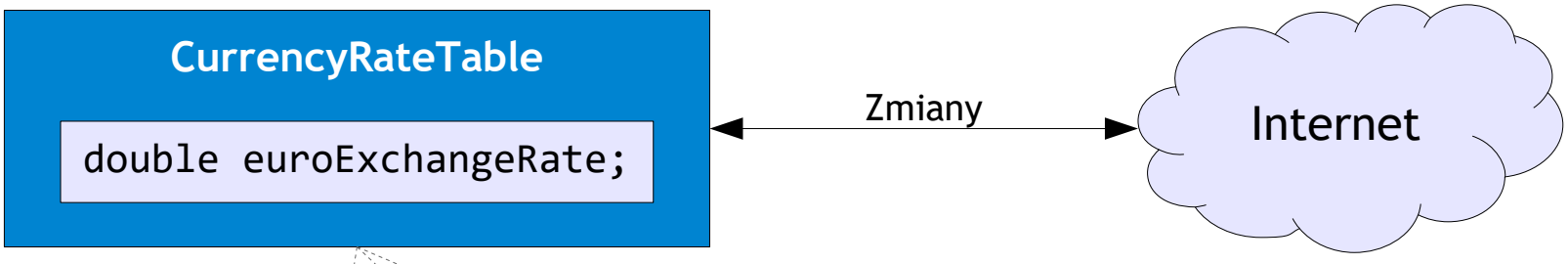
Podmiot powinien mieć możliwość powiadamiania zainteresowanych obiektów o zmianach, jednak związki pomiędzy klasami obiektów powinny być maksymalnie „luźne”



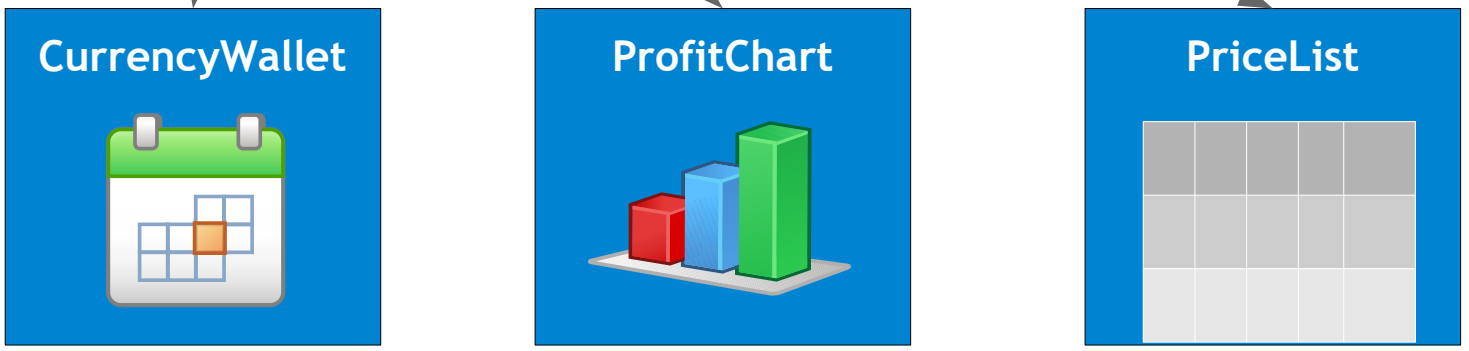
Obiekty, dla których zmiana stanu podmiotu jest istotna, powinny zostać o niej poinformowane - to **obserwatorzy** (**observers**)

Obserwator - koncepcja

Obiekt ***podmiotu*** (***subject***) zawierający informację, której zmiana powinna być rozgłoszona



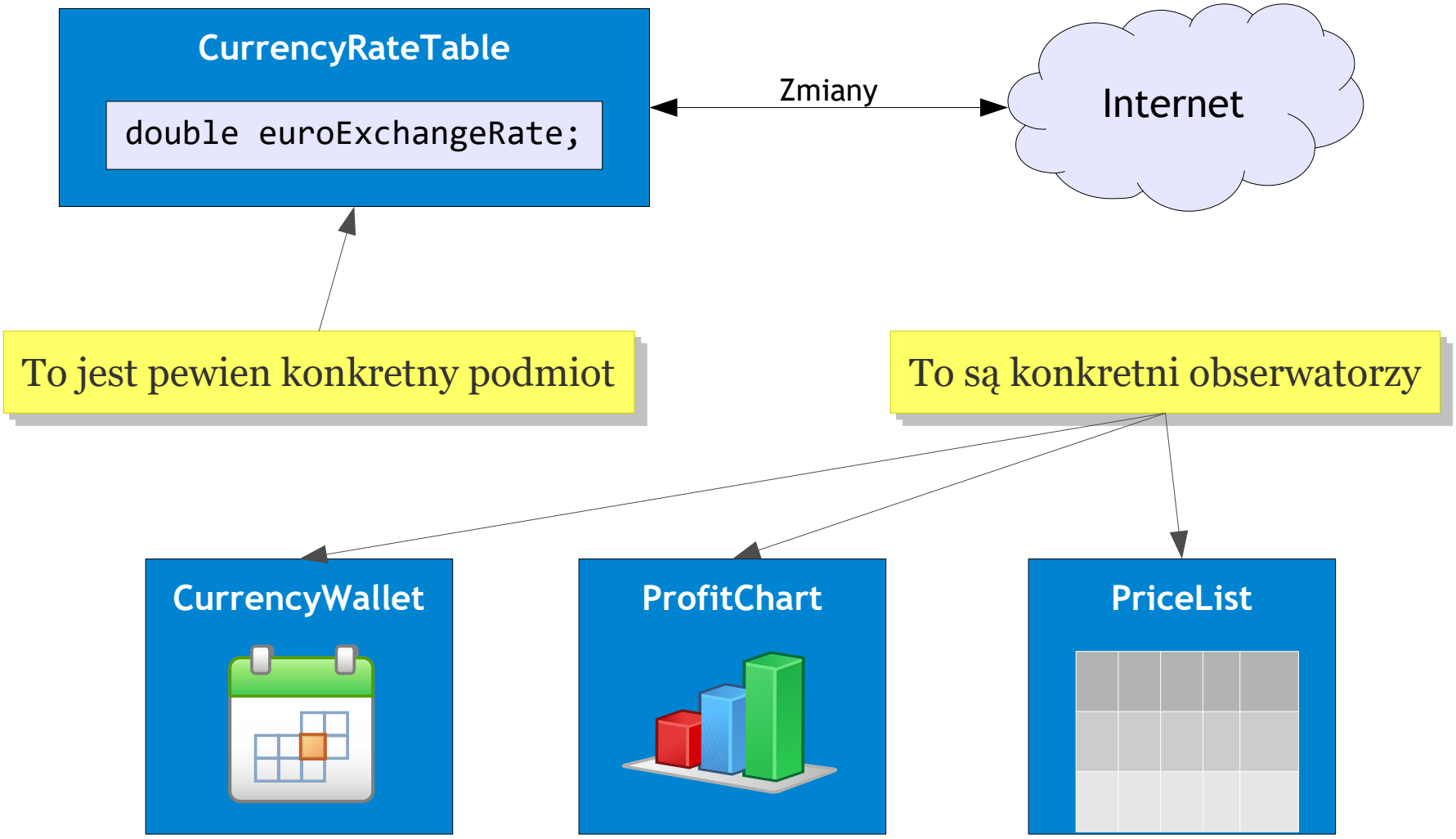
Powiązania obiektami powinny być tworzone dynamicznie, obiekty powinny jak najmniej wiedzieć o sobie, szczególnie obserwatorzy



Obiekty, dla których zmiana stanu podmiotu jest istotna, powinny zostać o niej poinformowane - to **obserwatorzy** (**observers**)

Obserwator - koncepcja

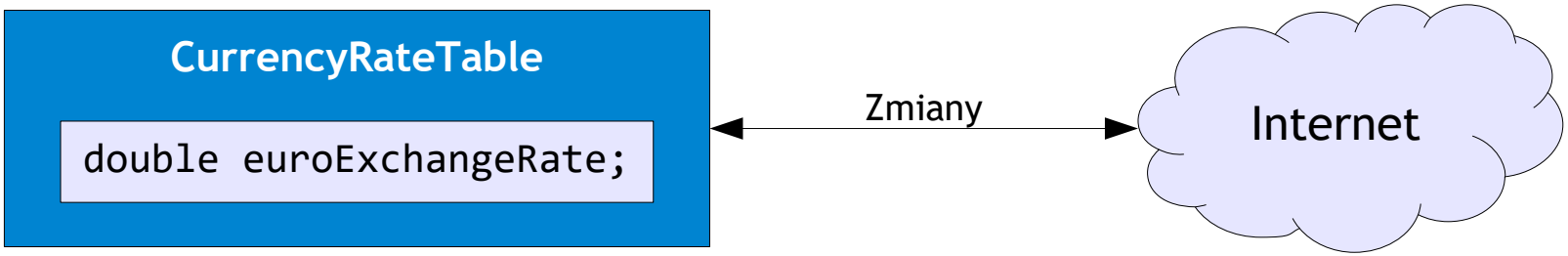
Obiekt ***podmiotu*** (***subject***) zawierający informację, której zmiana powinna być rozgłoszona



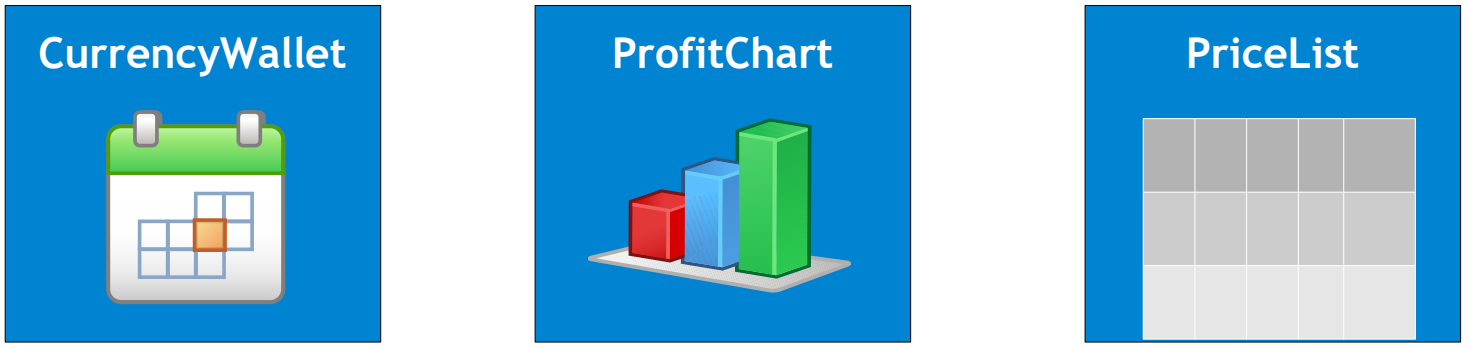
Obiekty, dla których zmiana stanu podmiotu jest istotna, powinny zostać o niej poinformowane - to **obserwatorzy** (**observers**)

Obserwator - koncepcja

Obiekt ***podmiotu*** (***subject***) zawierający informację, której zmiana powinna być rozgłoszona



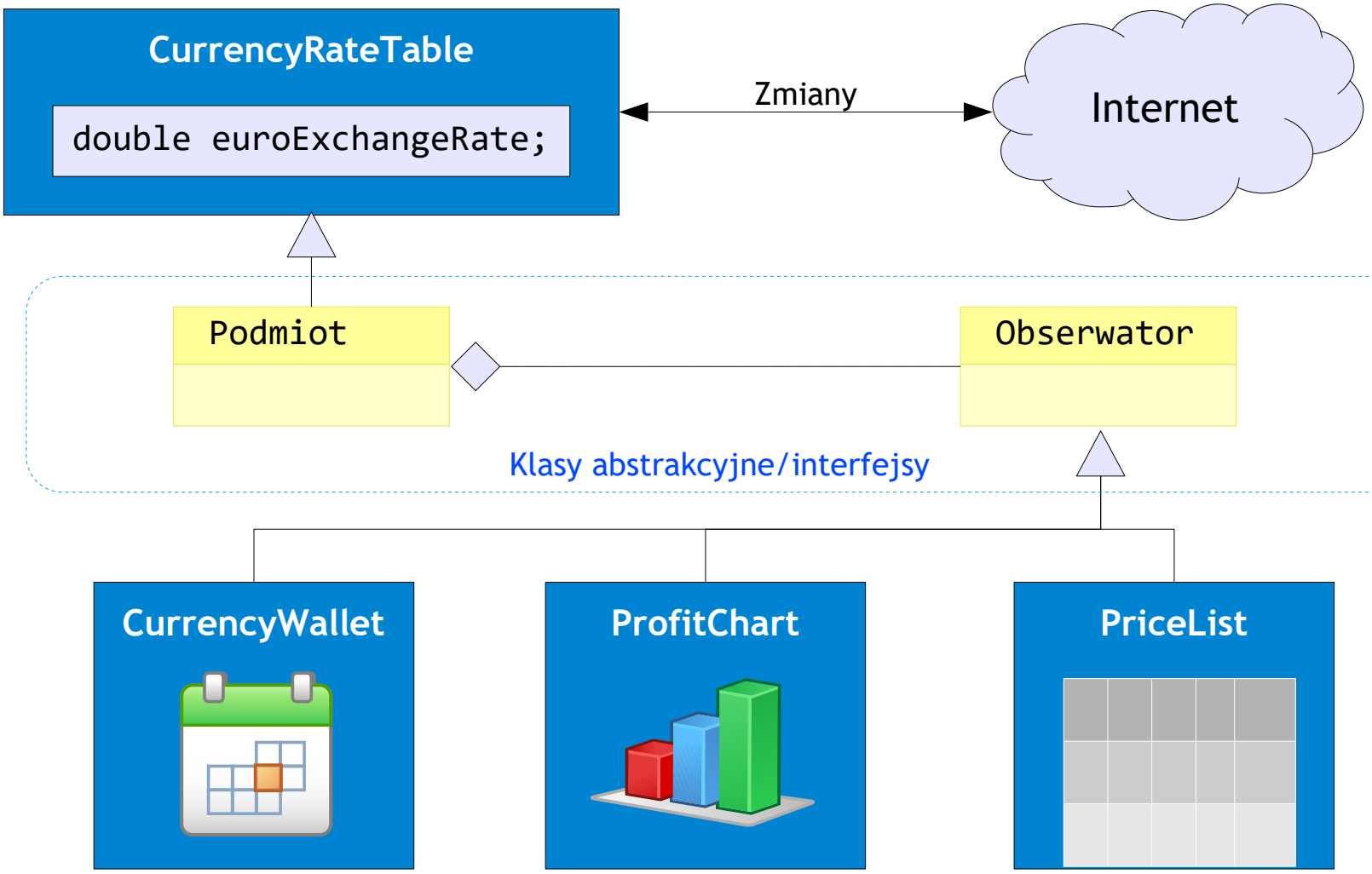
Wzorce projektowe dostarczają rozwiązania oderwanego od konkretów, dlatego przechodzimy na poziom wykorzystujący klasy abstrakcyjne i/lub interfejsy



Obiekty, dla których zmiana stanu podmiotu jest istotna, powinny zostać o niej poinformowane - to **obserwatorzy** (**observers**)

Obserwator - koncepcja

Obiekt ***podmiotu*** (***subject***) zawierający informację, której zmiana powinna być rozgłoszona

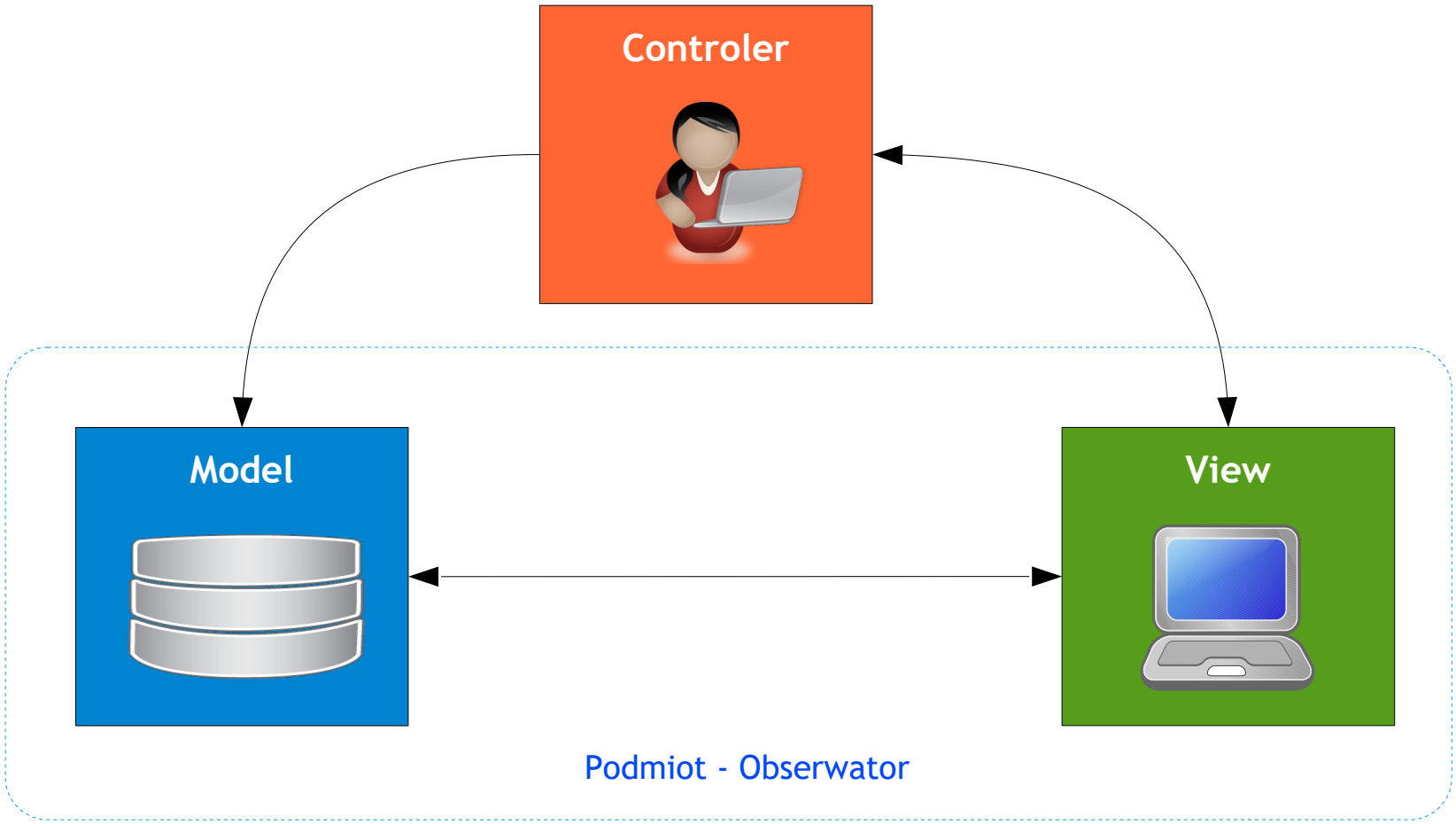


Obiekty, dla których zmiana stanu podmiotu jest istotna, powinny zostać o niej poinformowane - to **obserwatorzy** (**observers**)

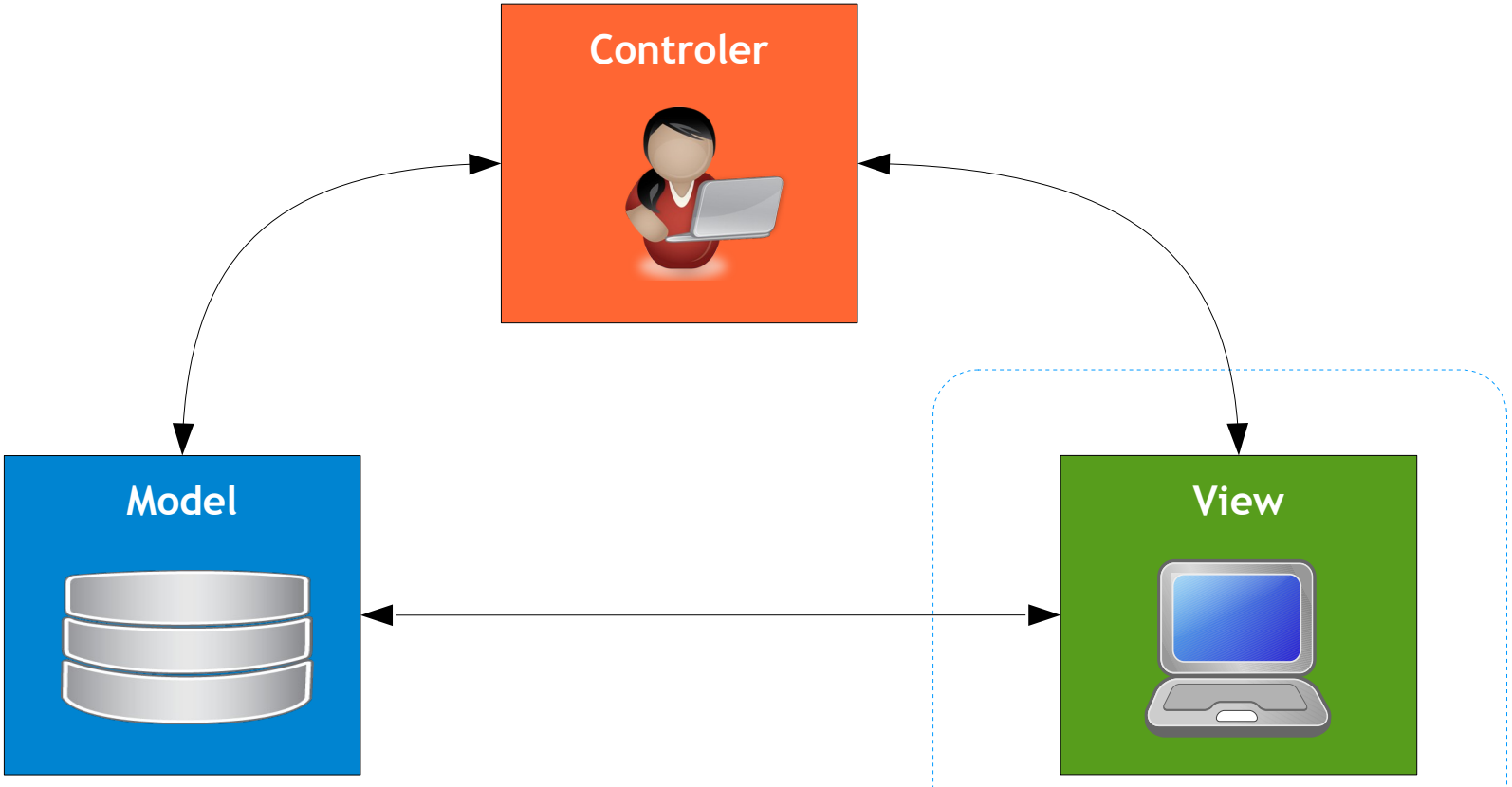
Obserwator

- Wzorzec ***Obserwator*** pozwala na nawiązanie luźnego powiązania pomiędzy *obiektem-podmiotem* a *obiektami-obserwatorami*.
- Obserwatorzy nie wiedzą o swoim istnieniu, podmiot jest powiązany z potencjalnie wieloma niezależnymi obserwatorami (typowo 1:N).
- Podmiot i obserwatorzy są powiązani poprzez abstrakcyjną specyfikację interfejsów/klas.

Obserwator a Model-View-Controler

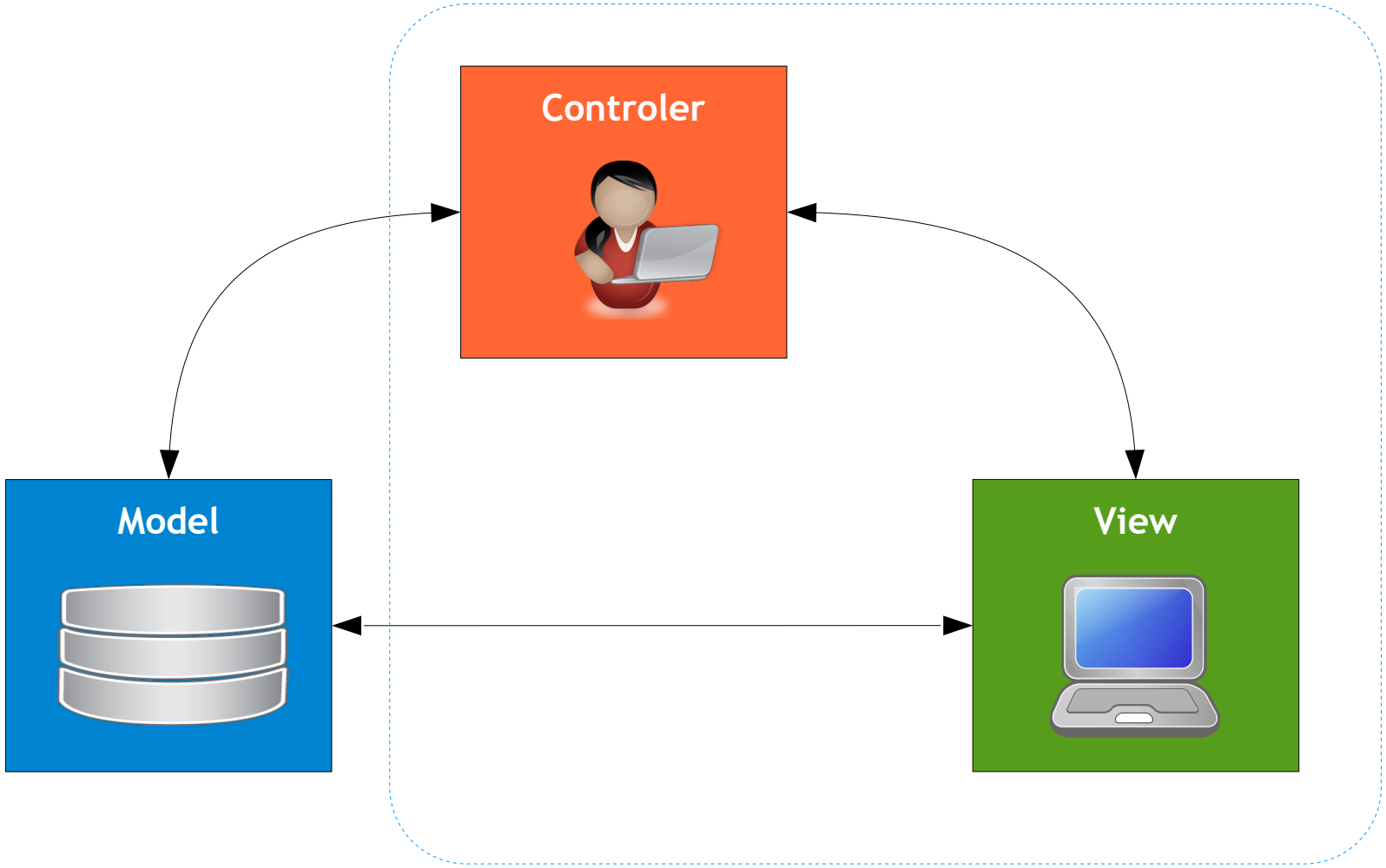


Model-View-Controler a Design Patterns



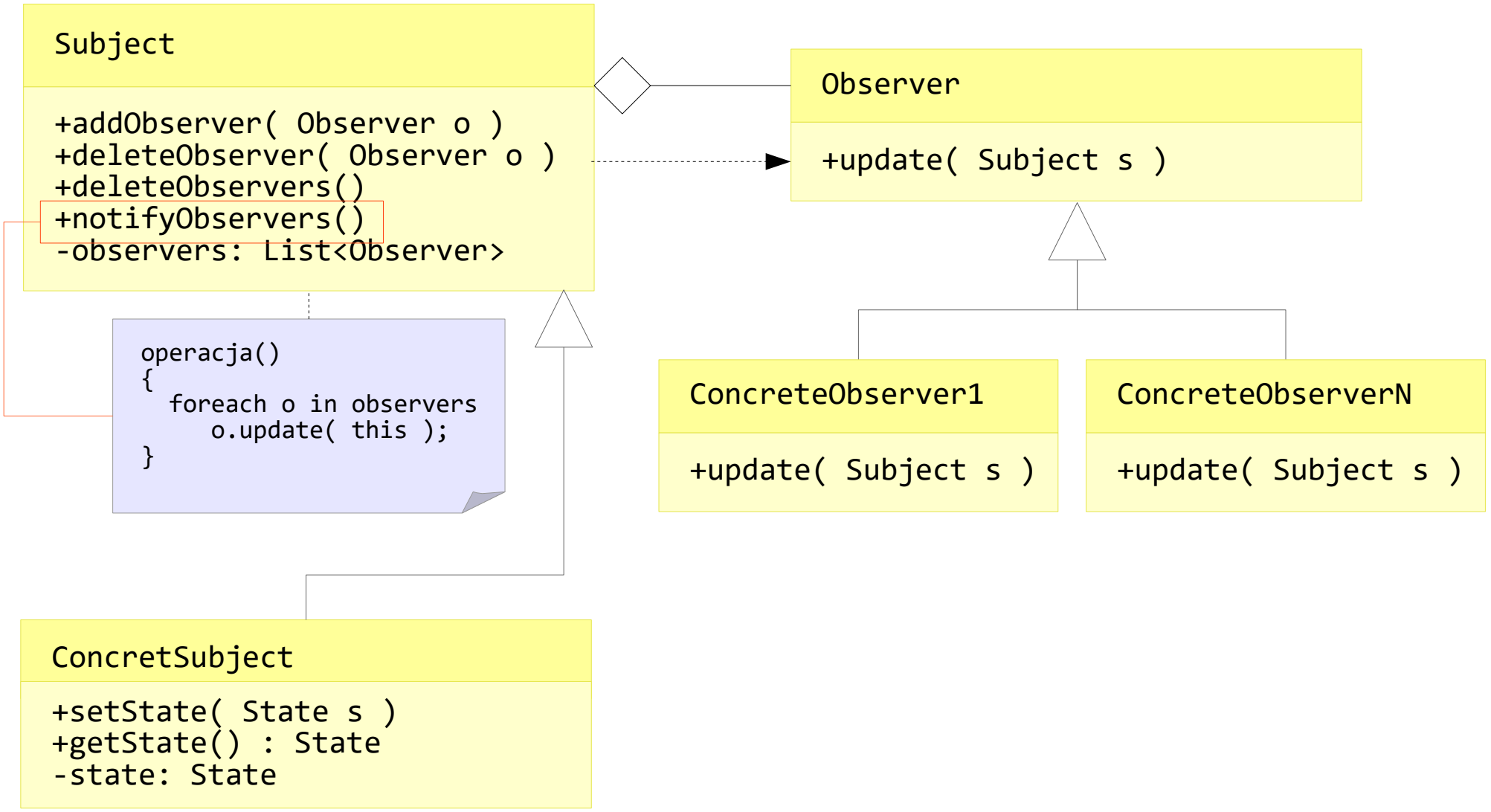
Kompozyt

Model-View-Controler a Design Patterns

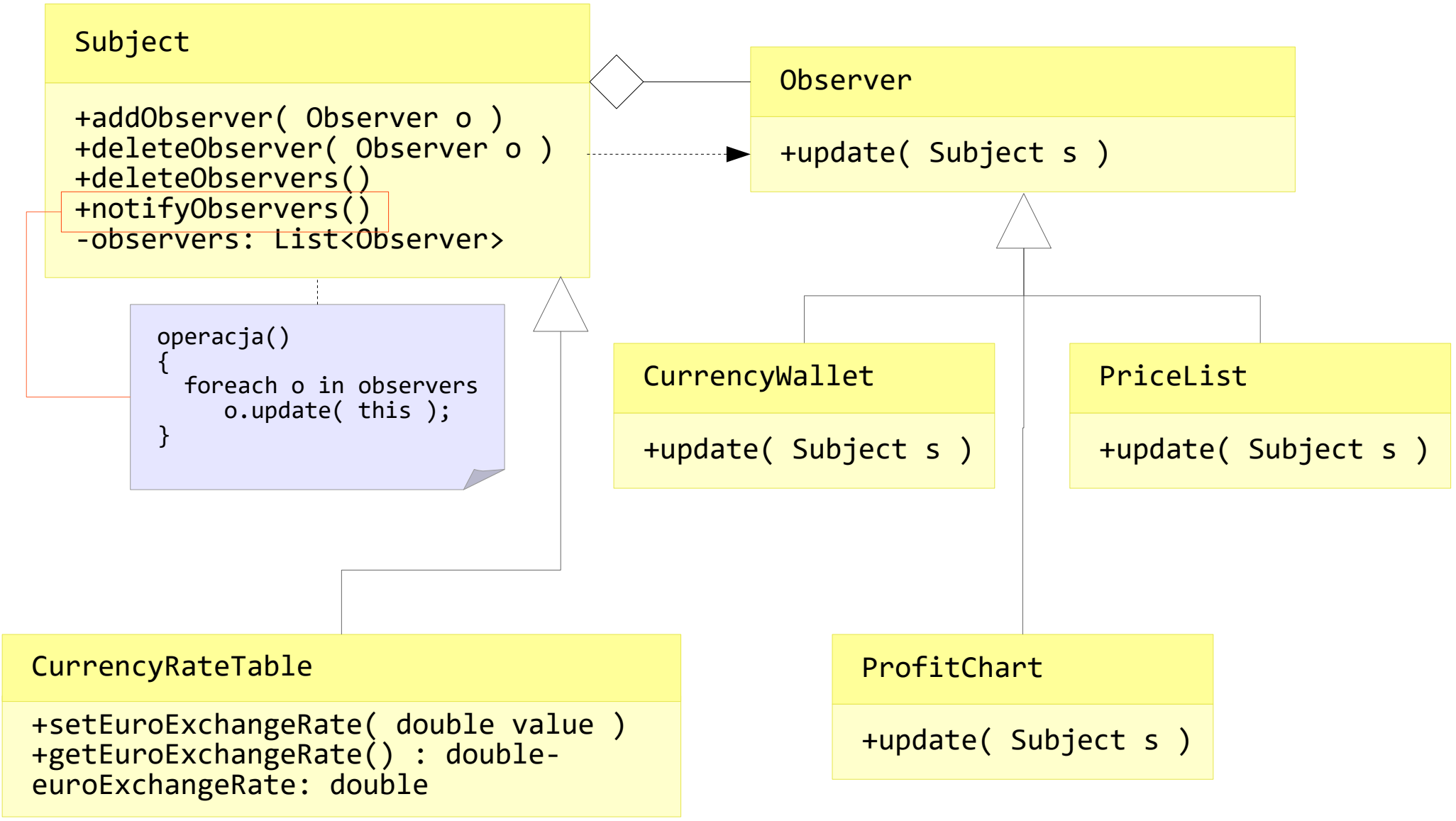


Strategia

Obserwator, ogólny schemat UML



Obserwator, schemat UML, przykład



Przykładowa implementacja obserwatora w języku Java

Elementy abstrakcyjne, obserwator, pierwsza przymiarka

```
interface Observer
{
    public void update( Subject s );
}
```

Elementy abstrakcyjne, podmiot

```
class Subject
{
    public void addObserver( Observer o )
    {
        observers.add( o );
    }

    public void deleteObserver( Observer o )
    {
        observers.remove( o );
    }

    public void deleteObservers()
    {
        observers.clear();
    }

    public void notifyObservers()
    {
        for( Observer o : observers )
            o.update( this );
    }

    protected ArrayList<Observer> observers = new ArrayList();
}
```

Elementy konkretne, podmiot

```
class CurrencyRateTable extends Subject
{
    CurrencyRateTable( double value )
    {
        euroExchangeRate = value;

    }

    public void setEuroExchangeRate( double value )
    {
       euroExchangeRate = value;
       notifyObservers();
    }

    public double getEuroExchangeRate()
    {
       return euroExchangeRate;
    }

    private double euroExchangeRate;
}
```

Elementy konkretne, obserwatorzy, problem

```
class CurrencyWallet implements Observer {

    public void update( Subject s )
    {
        System.out.println( "CurrencyWallet recived: " + getEuroExchangeRate ? );
    }
}

class ProfitChart implements Observer {
    public void update( Subject s )
    {
        System.out.println( "ProfitChart recived: " + getEuroExchangeRate ? );
    }
}

class PriceList implements Observer {
    public void update( Subject s )
    {
        System.out.println( "PriceList recived: " + getEuroExchangeRate ? );
    }
}
```

Elementy konkretne, obserwatorzy, nieeleganckie rozwiązanie

```
class CurrencyWallet implements Observer {

    public void update( Subject s )
    {
        System.out.println( "CurrencyWallet recived: " +
                            ((CurrencyRateTable)s).getEuroExchangeRate() );
    }
}

class ProfitChart implements Observer {
    public void update( Subject s )
    {
        System.out.println( "ProfitChart recived: " +
                            ((CurrencyRateTable)s).getEuroExchangeRate() );
    }
}

class PriceList implements Observer {
    public void update( Subject s )
    {
        System.out.println( "PriceList recived: " +
                            ((CurrencyRateTable)s).getEuroExchangeRate() );
    }
}
```

Elementy konkretne, obserwatorzy, lepsze rozwiązanie

```
class CurrencyWallet implements Observer
{
    CurrencyWallet( CurrencyRateTable s )
    {
        subject = s;
    }


    public void update( Subject s )
    {
        if( s == subject )
            System.out.println( "CurrencyWallet recived: " +
                                subject.getEuroExchangeRate() );
    }
    private CurrencyRateTable subject = null;
}
```

Przykład wykorzystania

```
CurrencyRateTable table = new CurrencyRateTable( 4.30 );

PriceList pl = new PriceList( table );
ProfitChart pc = new ProfitChart( table );
CurrencyWallet cw = new CurrencyWallet( table );

table.addObserver( pl );
table.addObserver( pc );
table.addObserver( cw );

table.setEuroExchangeRate( 4.50 );
table.setEuroExchangeRate( 4.20 );
table.setEuroExchangeRate( 4.30 );
```

```
PriceList recived: 4.5
ProfitChart recived: 4.5
CurrencyWallet recived: 4.5
PriceList recived: 4.2
ProfitChart recived: 4.2
CurrencyWallet recived: 4.2
PriceList recived: 4.3
ProfitChart recived: 4.3
CurrencyWallet recived: 4.3
```

Observable - Java

Constructor Summary

Constructors

Constructor and Description
`Observable()` Construct an Observable with zero Observers.

Method Summary

Methods

Modifier and Type	Method and Description
`void`	`addObserver(Observer o)` Adds an observer to the set of observers for this object, provided that it is not the same
`protected void`	`clearChanged()` Indicates that this object has no longer changed, or that it has already notified all of its
`int`	`countObservers()` Returns the number of observers of this `Observable` object.
`void`	`deleteObserver(Observer o)` Deletes an observer from the set of observers of this object.
`void`	`deleteObservers()` Clears the observer list so that this object no longer has any observers.
`boolean`	`hasChanged()` Tests if this object has changed.
`void`	`notifyObservers()`

Observer - Java

Interface Observer

```
public interface Observer
```

A class can implement the `Observer` interface when it wants to be informed of changes in observable objects.

Since:

JDK1.0

See Also:

`Observable`

Method Summary

Methods

Modifier and Type	Method and Description
`void`	`update(Observable o, Object arg)` This method is called whenever the observed object is changed.

Method Detail

update

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

This method is called whenever the observed object is changed. An application calls an `Observable` object's `notifyObservers` method to have all the object's

Parameters: