Operatori, delegati i događaji

Programiranje korisničkih interfejsa

Bojan Furlan

Operators and Methods

- **Using methods**
  - Reduces clarity
  - Increases risk of errors, both syntactic and semantic

```
Complex Var1 = Complex.Add(Var2,
      Complex.Add(Complex .Add(Var3, Var4), Var5));
```

- **Using operators**
  - Makes expressions clear

```
Complex Var1 = Var2 + Var3 + Var4 + Var5;
```

Introduction to Operator Overloading

- **Operator overloading**
  - Define your own operators only when appropriate
- **Operator syntax**
  - Operator**op**, where **op** is the operator being overloaded
- **Example**

```
public static Time operator+(Time t1, Time t2)
{
      int newHours = t1.hours + t2.hours;
      int newMinutes = t1.minutes + t2.minutes;
      return new Time(newHours, newMinutes);
}
```

Overloading Relational Operators

- **Relational operators must be paired**
  - < and >
  - <= and >=
  - == and !=
- **For consistency:**
  - Override the Equals method if overloading == and !=
  - Override the GetHashCode method if overriding equals method

Overloading Conversion Operators

- **Overloaded conversion operators**

```
public static explicit operator Time (float hours)
{ ... }
public static explicit operator float (Time t1)
{ ... }
public static implicit operator string (Time t1)
{ ... }
```

- **If a class defines a string conversion operator**
  - The class should override ToString

Overloading Operators Multiple Times

- The same operator can be overloaded multiple times

```
public static Time operator+(Time t1, int hours)
{...}

public static Time operator+(Time t1, float hours)
{...}

public static Time operator-(Time t1, int hours)
{...}

public static Time operator-(Time t1, float hours)
{...}
```

Delegati

Scenario: Power Station

- **The problem**
  - How to respond to temperature events in a power station
  - Specifically, if the temperature of the reactor core rises above a certain temperature, coolant pumps need to be alerted and switched on
- **Possible solutions**
  - Should all coolant pumps monitor the core temperature?
  - Should a component that monitors the core turn on the appropriate pumps when the temperature changes?

Analyzing the Problem

- **Existing concerns**
  - There may be several types of pumps, supplied by different manufacturers
  - Each pump could have its own method for activation (driver)
- **Future concerns**
  - To add a new pump, the entire code will need to change
  - A high overhead cost will result with every such addition

Različiti drajveri za pumpu

```
public class ElectricPumpDriver
{
...
        public void StartElectricPumpRunning( )
        {
        ...
        }
}
public class PneumaticPumpDriver
{
...
        public void SwitchOn( )
        {
        ...
        }
}
```

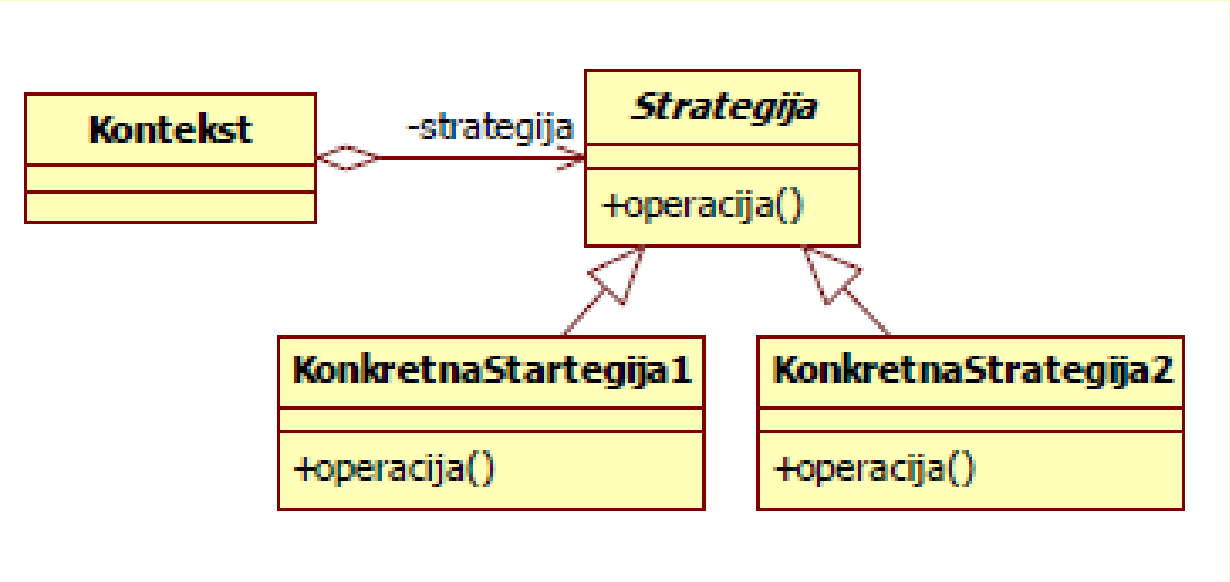
Kako uključiti sve pumpe

```
public class CoreTempMonitor
{
        private ArrayList pumps = new ArrayList();

        public void Add(object pump)
        {
                pumps.Add(pump);
        }
        public void SwitchOnAllPumps()
        {
                foreach (object pump in pumps) {
                        if (pump is ElectricPumpDriver) {
                           ((ElectricPumpDriver)pump).StartElectricPumpRunning();
                        }
                        if (pump is PneumaticPumpDriver) {
                            ((PneumaticPumpDriver)pump).SwitchOn();
                        }
                ...
                }
       }
}
```

Jedan primer rešenja

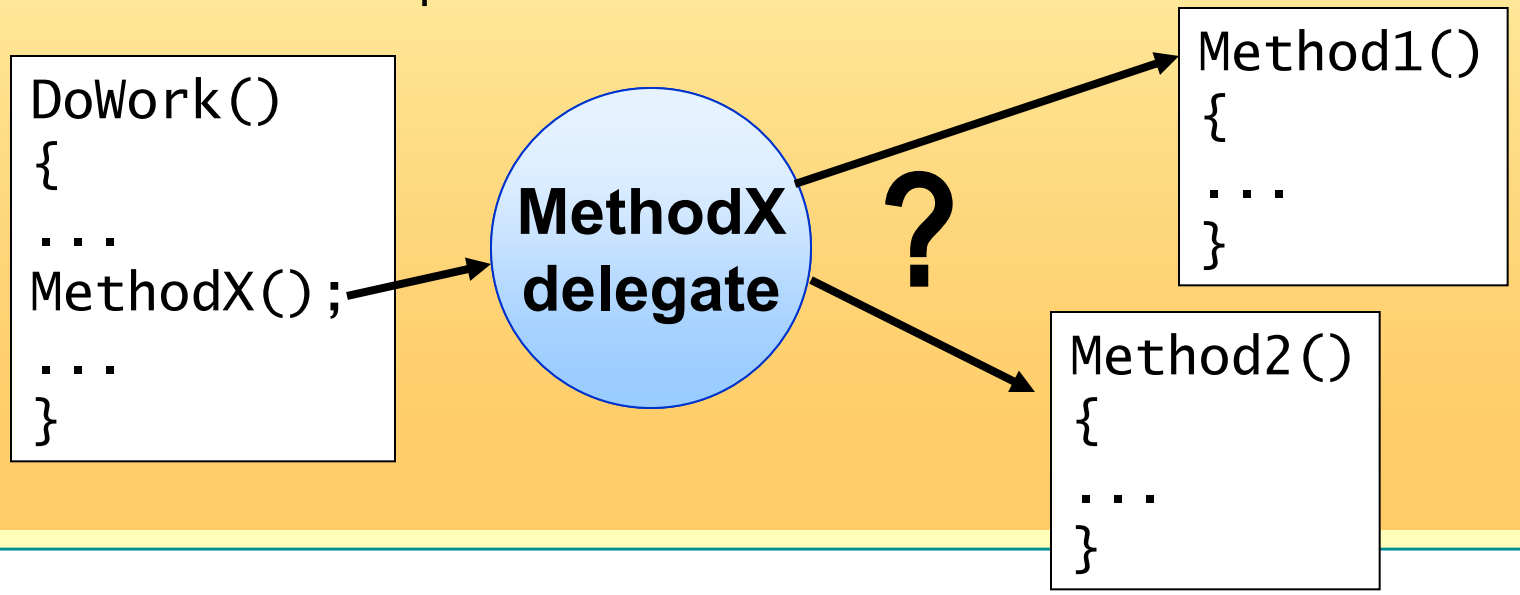
- Implementirati projektni uzorak Strategija
- U svakoj konkretnoj klasi dodati kod koji poziva drajver specifične pumpe



- Rešen problem dodavanja novog drajvera bez menjanja celog koda, ali ne i problem “glue” koda

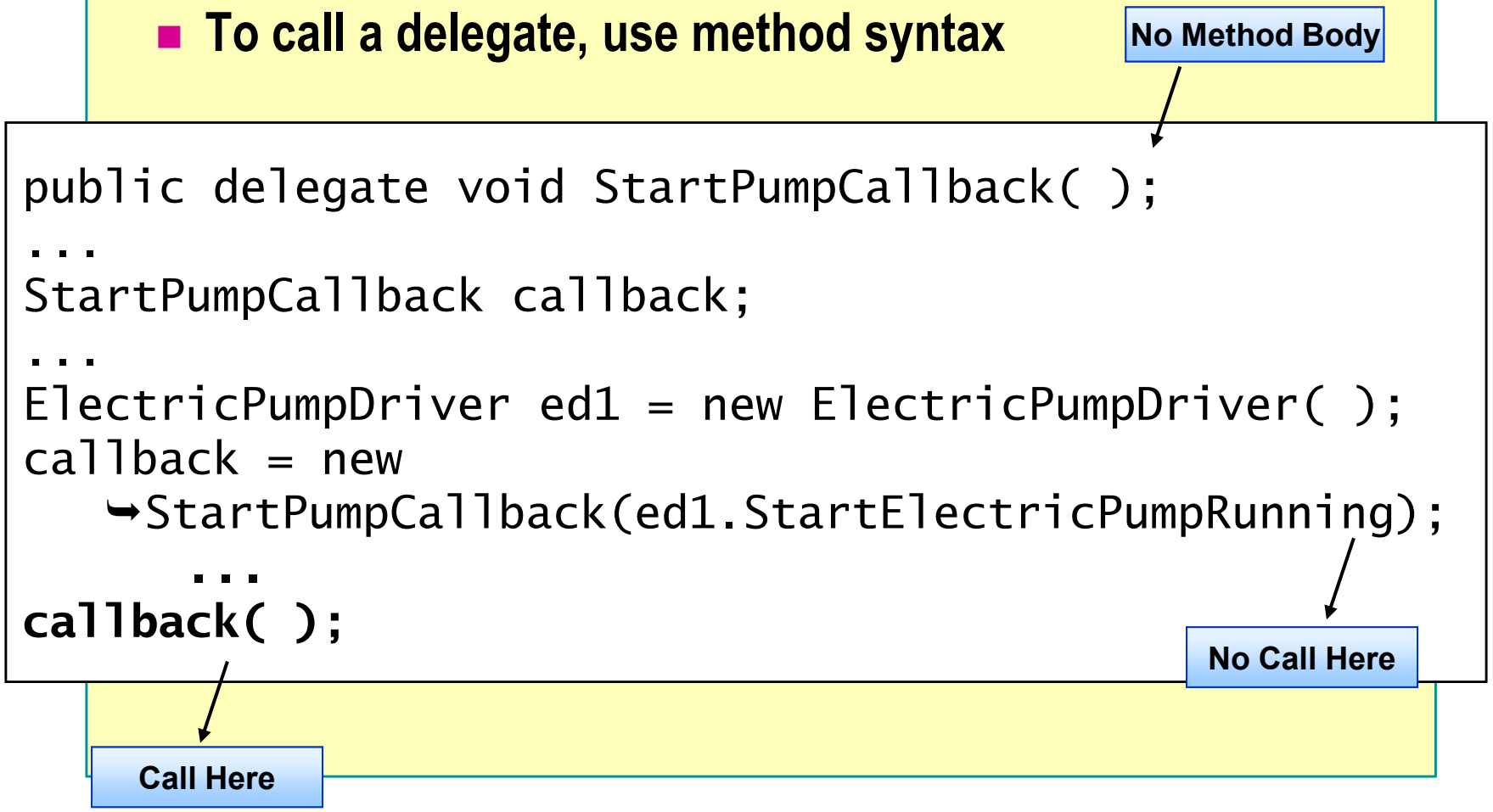
Creating Delegates

- **A delegate allows a method to be called indirectly**
  - It contains a reference to a method
  - All methods invoked by the same delegate must have the same parameters and return value



Using Delegates

- To call a delegate, use method syntax



Kako uključiti sve pumpe

```
public class CoreTempMonitor
{
        private ArrayList callbacks = new ArrayList( );

        public void Add(StartPumpCallback callback)
        {
                callbacks.Add(callback);
        }

        public void SwitchOnAllPumps( )
        {
                foreach(StartPumpCallback callback in callbacks)
                {
                        callback( );
                }
        }
}
```

Delegate vs. Interface

- **Use a delegate in the following circumstances:**
  - An eventing design pattern is used.
  - It is desirable to encapsulate a static method.
  - The caller has no need to access other properties, methods, or interfaces on the object implementing the method.
  - Easy composition is desired.
  - Methods may be named differently

Delegate vs. Interface

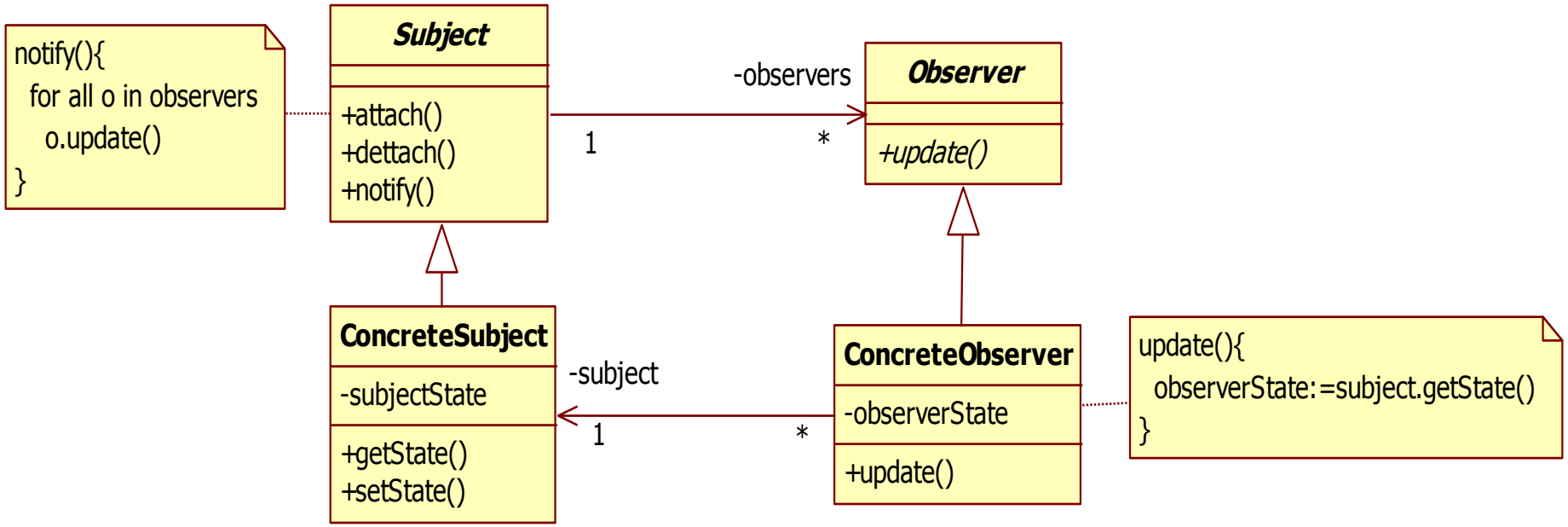
- **Use an interface in the following circumstances:**
  - There is a group of related methods that may be called.
  - The class using the interface will want to cast that interface to other interface or class types.
  - The method being implemented is linked to the type or identity of the class: for example, comparison methods.

Događaji

How Events Work

- **Publisher**
  - Raises an event to alert all interested objects (subscribers)
- **Subscriber**
  - Provides a method to be called when the event is raised

Posmatrač

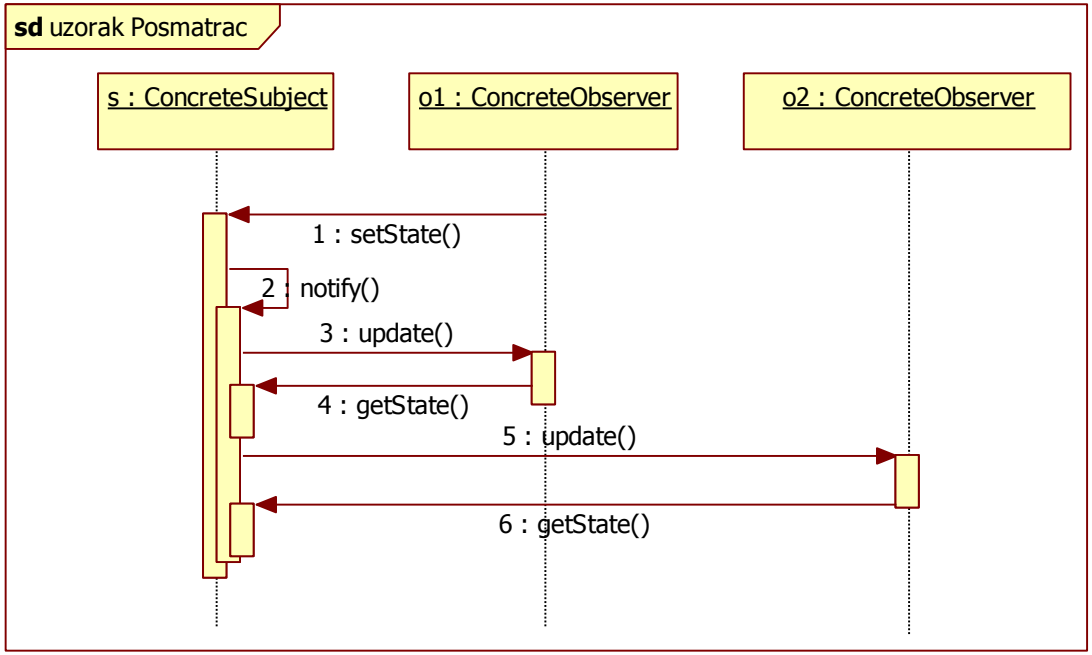

notify(){
for all o in observers
o.update()
}
Subject
+attach()
+dettach()
+notify()
-observers
1
*
Observer
+update()
ConcreteSubject
-subjectState
+getState()
+setState()
-subject
1
*
ConcreteObserver
-observerState
+update()
update(){
observerState:=subject.getState()
}

Posmatrač

- Participanti:
  - `Subject` klasa
    - zna svoje posmatrače; proizvoljan broj posmatrača može da nadgleda subjekat
    - obezbeđuje interfejs za pridruživanje i razdruživanje posmatrača
  - `Observer` klasa
    - definiše interfejs za modifikovanje objekata kojima se signalizira promena subjekta
  - `ConcreteSubject` klasa
    - čuva stanje od interesa za konkretne posmatrače
    - šalje signal posmatračima kada se promeni stanje
  - `ConcreteObserver` klasa
    - poseduje referencu na konkretan subjekat
    - čuva stanje koje treba da bude u konzistenciji sa stanjem konkretnog subjekta
    - implementira interfejs za promenu sopstvenog stanja da očuva konzistenciju sa subjektom

Posmatrač

- Kolaboracije:
  - konkretan subjekat signalizira svojim posmatračima svaku promenu svog stanja
  - nakon poziva `update`, konkretan posmatrač traži od subjekta informaciju o stanju
  - posmatrač koristi informaciju o stanju subjekta da ažurira svoje stanje



Defining Events

- **Defining an event**

```
public delegate void StartPumpCallback();
private event StartPumpCallback CoreOverheating;
```

- **Subscribing to an event**

```
PneumaticPumpDriver pd1 = new PneumaticPumpDriver( );
...
CoreOverheating += new StartPumpCallback(pd1.SwitchOn);
```

- **Notifying subscribers to an event**

```
public void SwitchOnAllPumps( ) {
  if (CoreOverheating != null) {
      CoreOverheating( );
  }
}
```

Passing Event Parameters

- **Parameters for events should be passed as EventArgs**
  - Define a class descended from EventArgs to act as a container for event parameters
- **The same subscribing method may be called by several events**
  - Always pass the event publisher (sender) as the first parameter to the method

Primer

```
public class CoreOverheatingEventArgs: EventArgs
{
        private readonly int temperature;

        public CoreOverheatingEventArgs(int temperature)
        {
                this.temperature = temperature;
        }

        public int GetTemperature( )
        {
                return temperature;
        }
}
```

Primer

```
public class ElectricPumpDriver
{
        ...
        public void StartElectricPumpRunning(object sender,
        CoreOverheatingEventArgs args)
        {
                // Examine the temperature
                int currentTemperature = args.GetTemperature( );
                // Start the pump at the required speed for
                // this temperature
                ...
        }
        ...
}
```

Delegates vs. Events

- **Event = a modifier for Multicast Delegate**
  - event can be included in an interface declaration

```
public delegate void MsgHandler ( );

interface ITest
{
  event MsgHandler msgNotifier; // compiles
  MsgHandler msgNotifier2;  // error CS0525: Interfaces cannot contain fields
}
```

Delegates vs. Events

- **event can only be invoked from within the class that declared it, delegate field can be invoked by whoever has access to it**

```
public delegate void MsgHandler ();
class Class1  {
   public static event MsgHandler msgNotifier;
   public static MsgHandler msgNotifier2;

  }

  class Class2   {
   public void test(){
    Class1.msgNotifier();
              // error CS0070: The event 'Class1.msgNotifier'
              //can only appear on the left hand side of += or -=
              //(except when used from within the type 'Class1')
    Class1.msgNotifier2(); // compiles fine
   }
  …
```