Technologia Programowania 2017/2018
Wykład 7

Jakub Lemiesz

Plan kolejnych wykładów

28.11 - JDBC + Hibernate

05.12 - systemy kontroli wersji, Git

12.12 - model asynchroniczny + Reactor + Akka

19.12 - Akka + Play

09.01 - RegExp, testowanie

16.01 - kolokwium

23.01 - Spring (DI, AOP)

//25.01 - początek sesji

Kolkwium

- Kolokwium będzie 16 stycznia 2018 na wykładzie
- Piszemy w dwóch turach:
  1. 17:05-18:00 — nazwiska od A do Ł
  2. 18:05-19:00 — nazwiska od M do Ż
- Można mieć ze sobą jedną kartkę formatu A4 zapisaną/zadrukowaną dwustronnie i wyraźnie podpisaną

Co na kolokwium?

1. Wzorce projektowe GoF, UML !!
2. SOLID, GRASP
3. Polimorfizm, override vs. overload
4. Wyrażenia regularne, walidacja XML
5. Ogólna wiedza z wykładu
   (*np. MVC, dependency inversion, Akka, Agile,...*)

GoF:Command — czyli obiekty-polecenia

Chcemy kolejkować zadania by wykonać je np.
w ustalonym czasie, w odpowiedniej kolejności,
"wszystkie albo żadne"

Przykłady:

- akcje GUI, edycja dokumentu - polecenia można układać na stosie i cofać (*zadanie na ćwiczenia*)
- obsługa zdarzeń na serwerze: serwer odbiera żądania, tworzy z nich obiekty i określa priorytety
- transakcje bazodanowe

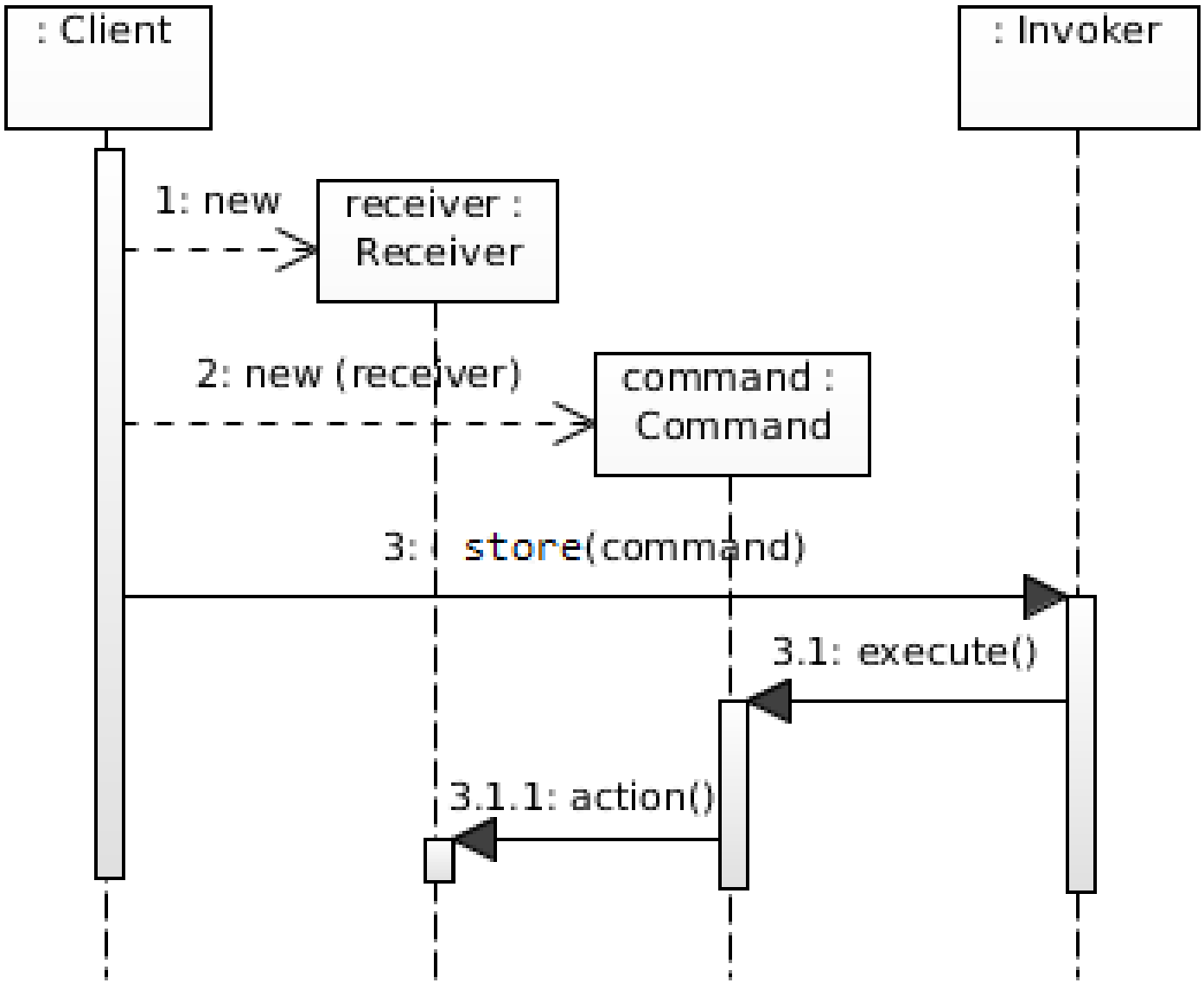
GoF:Command — idea

1. Zamykaj wszystkie pojawiające się polecenia w ramach obiektów-poleceń implementujących wspólny interfejs, np. w ICommand polimorficzna metoda execute(...)

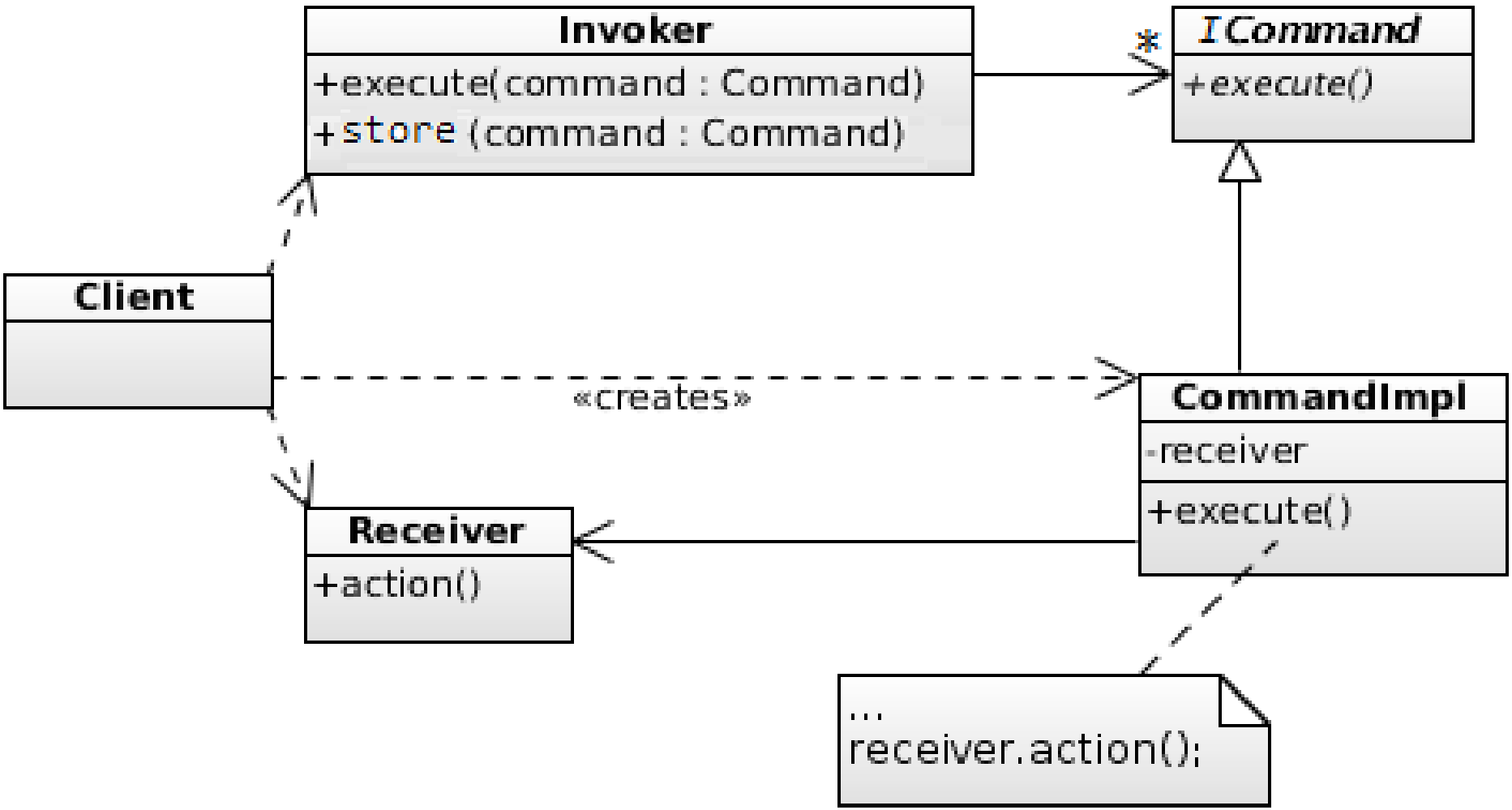
2. Stworzone obiekty-polecenia przekaż do specjalnej klasy Invoker odpowiedzialnej za kolejkowanie i wykonanie ich w odpowiednim czasie

Rróżnice między GoF:Command a GoF:Strategy?

GoF:Command – ogólny schemat



GoF:Command – ogólny schemat



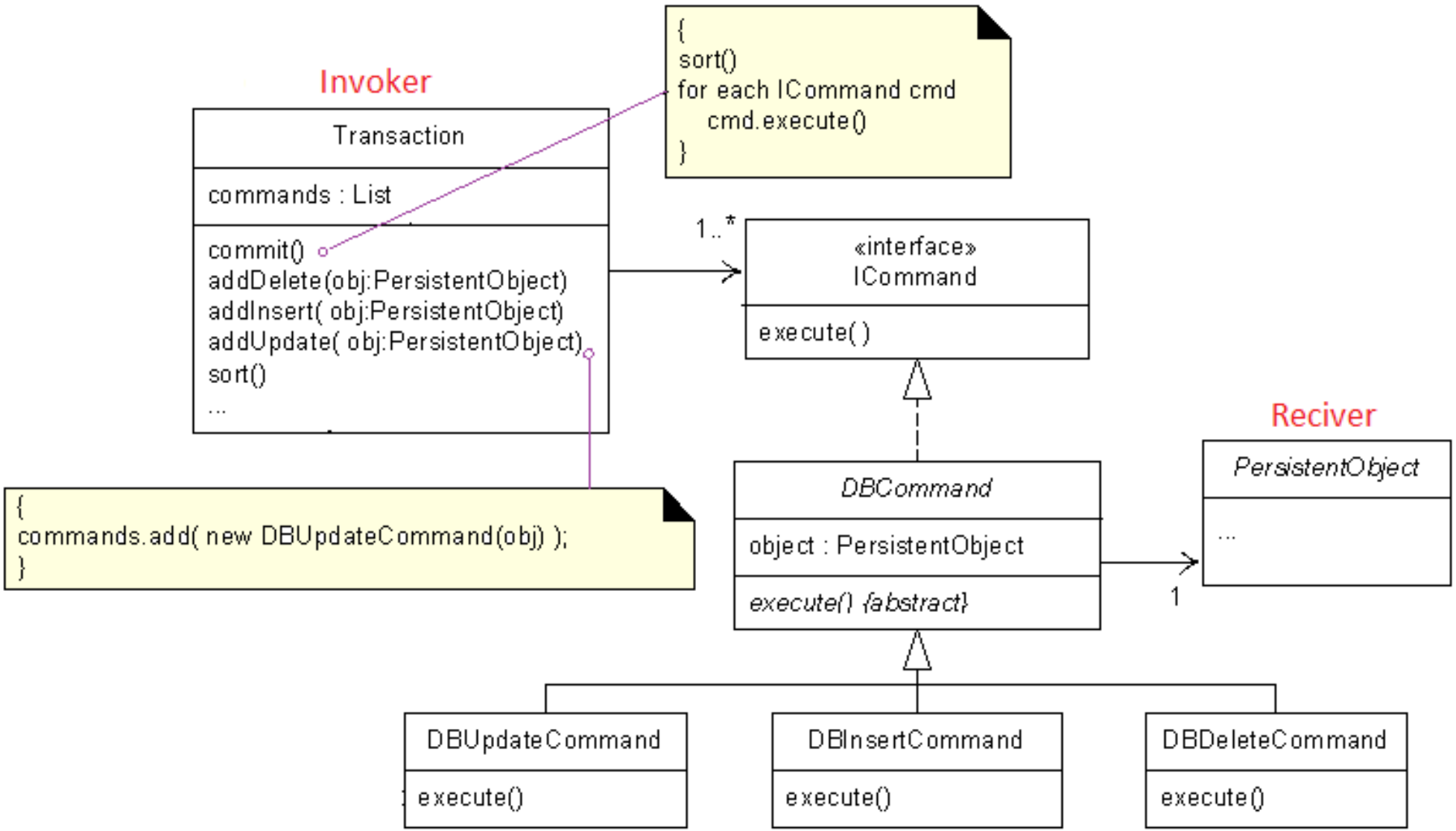
GoF:Command — przykład

Transakcje bazodanowe:

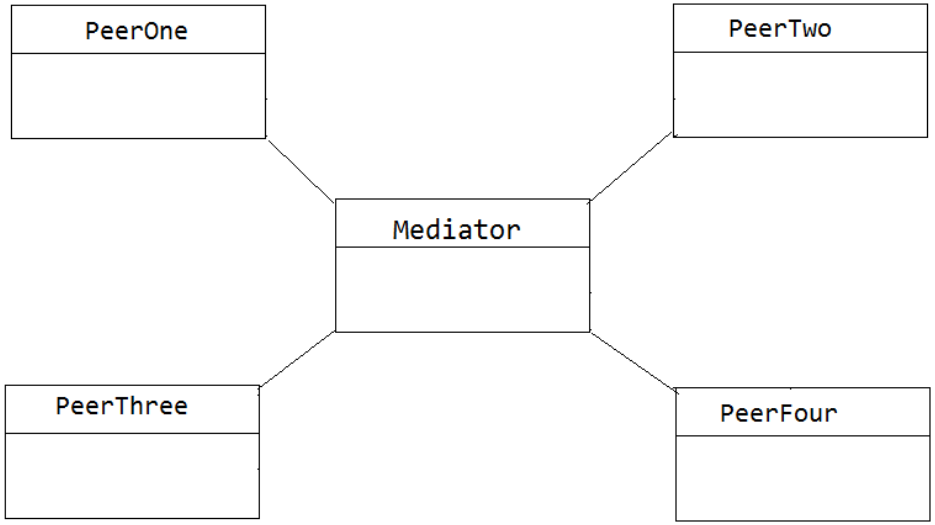
- Transakcja — zbiór poleceń insert, update, delete
- Transakcja — czynność atomowa, tzn. wszystkie zadania muszą być wykonane albo żadne z nich
- Kolejność poleceń istotna dla spójności danych: `inserts`, potem `updates`, potem `deletes`,

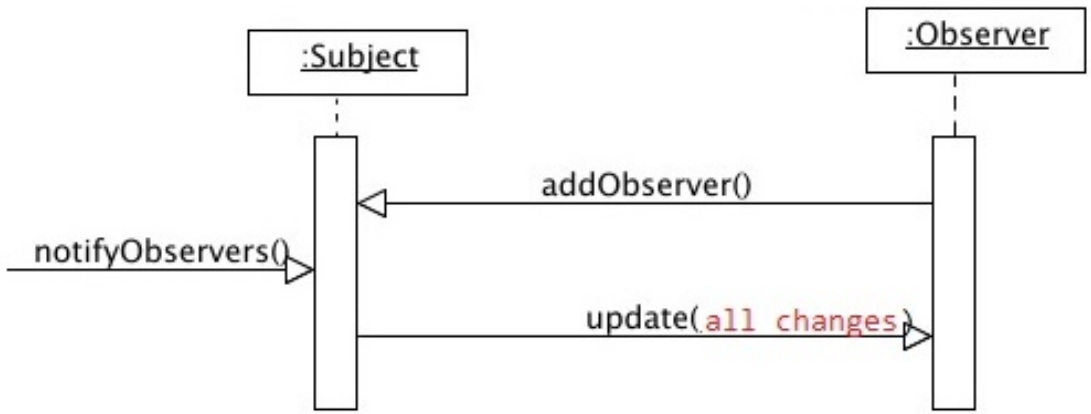
  Zobacz np.: http://stackoverflow.com/questions/12616336/how-is-hibernate-deciding-order-of-update-insert-delete

GoF:Command — przykład

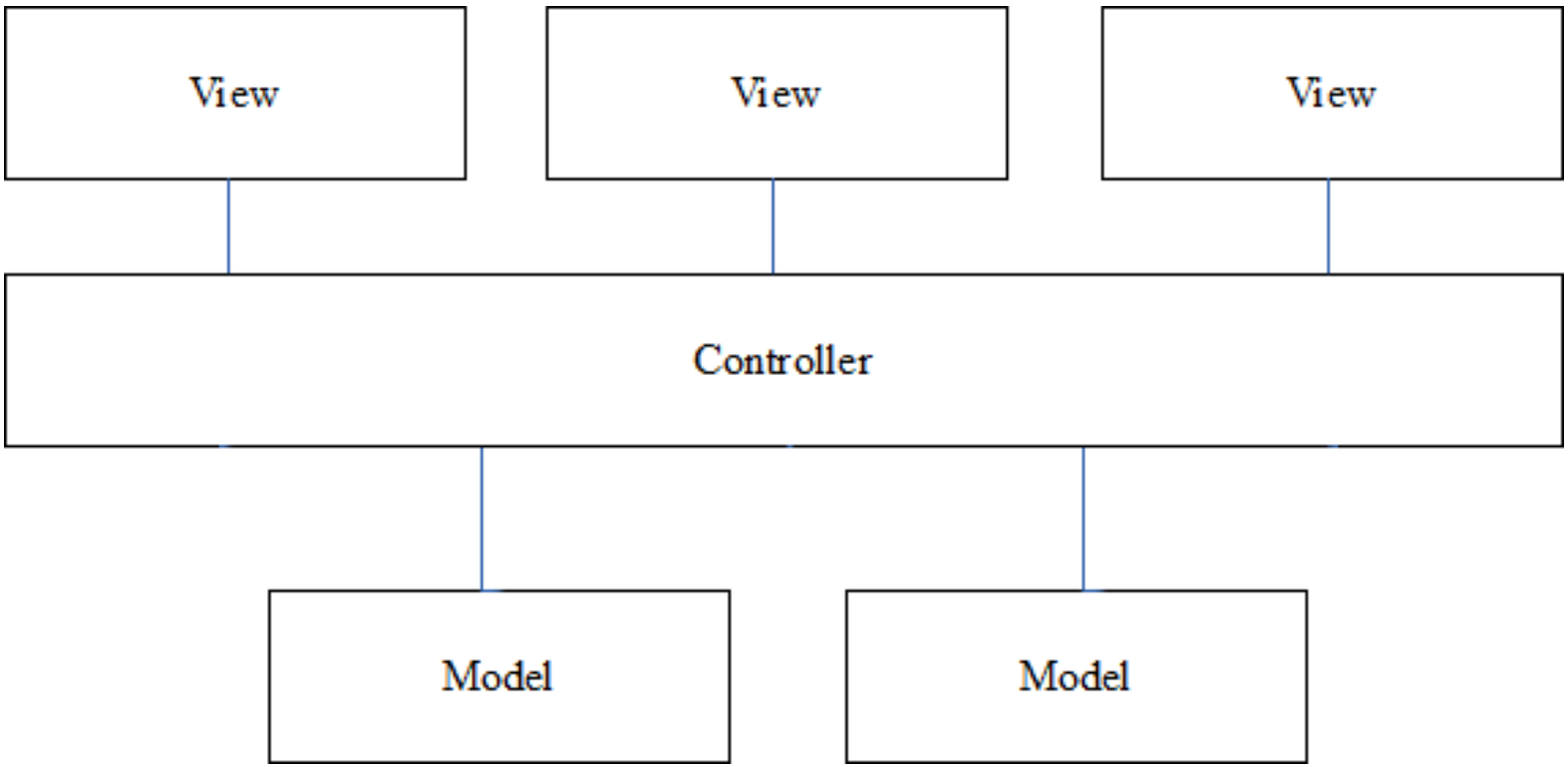


MVC ≈ Mediator + Observer

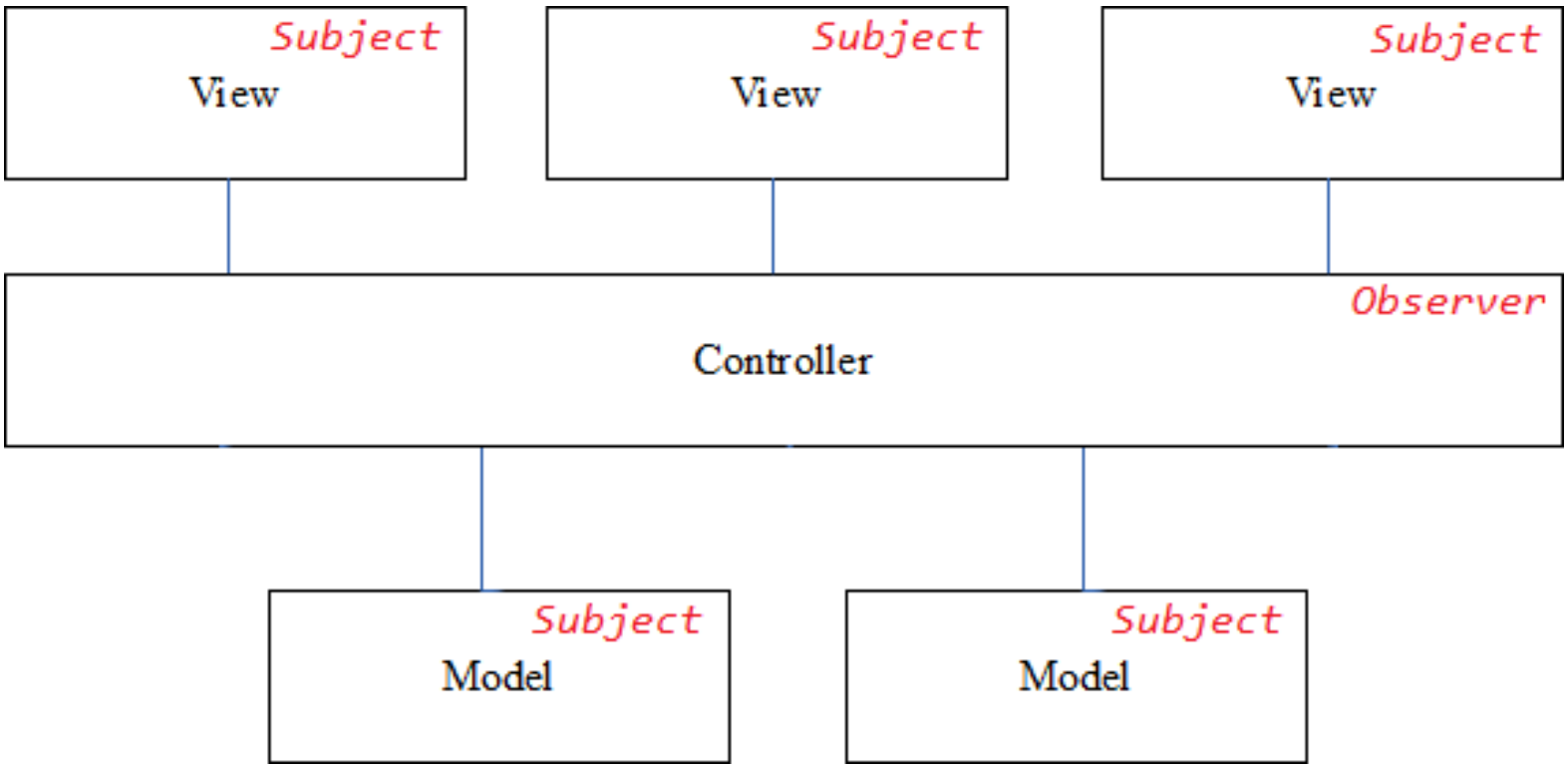




MVC ≈ Mediator + Observer



MVC ≈ Mediator + Observer



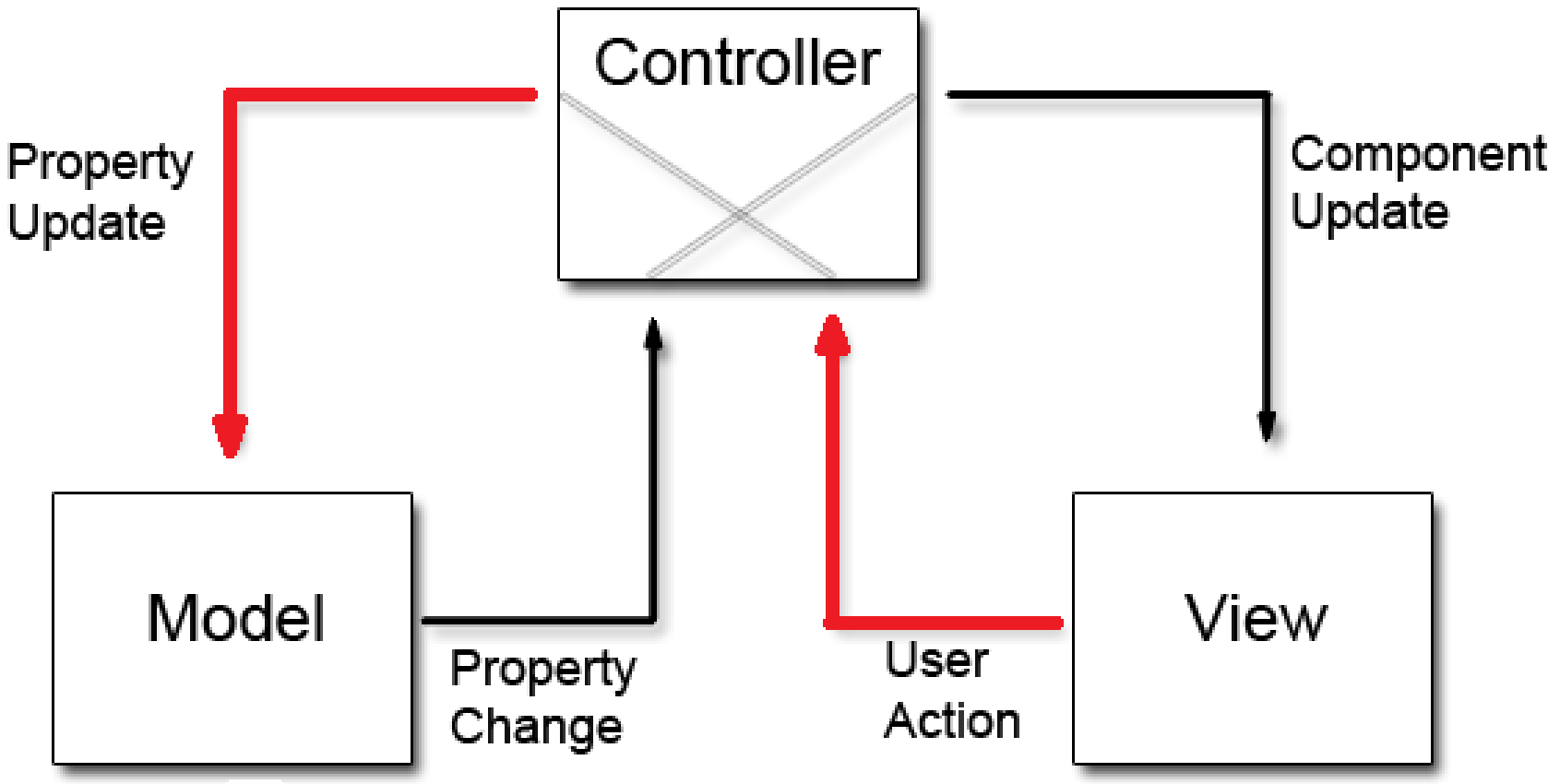
MVC – kontroler jako mediator

```
public class Controller implements PropertyChangeListe
{
    ArrayList<IView> registeredViews;
    ArrayList<AbstractModel> registeredModels;

    //foreach new model m
    registeredModels.add(m);
    m.addPropertyChangeListener(this);

    //foreach new view v
    registeredViews.add(v);
    v.add...Listener(this);//albo new View(controller)
```

MVC – zdarzenia w GUI



MVC – zdarzenia w GUI

```
//gdzieś w JFrame
import java.awt.event.*;
...

//nowy obserwator dla wybranej akcji
jButton.addActionListener(
      new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            controller.changeAge(jTextField.getText());
          }
       }
)
```

Gdzie jest Subject, Observer, addObserver(), update()?
Która klasa jest anonimowa?

MVC – aktualizacja modelu

```
//gdzieś w JFrame
controller.changeAge(jTextField.getText());

//gdzieś w kontrolerze
public void changeAge(String newValue){
    for (AbstractModel model: registeredModels) {
        model.setAge(newValue);
}
```

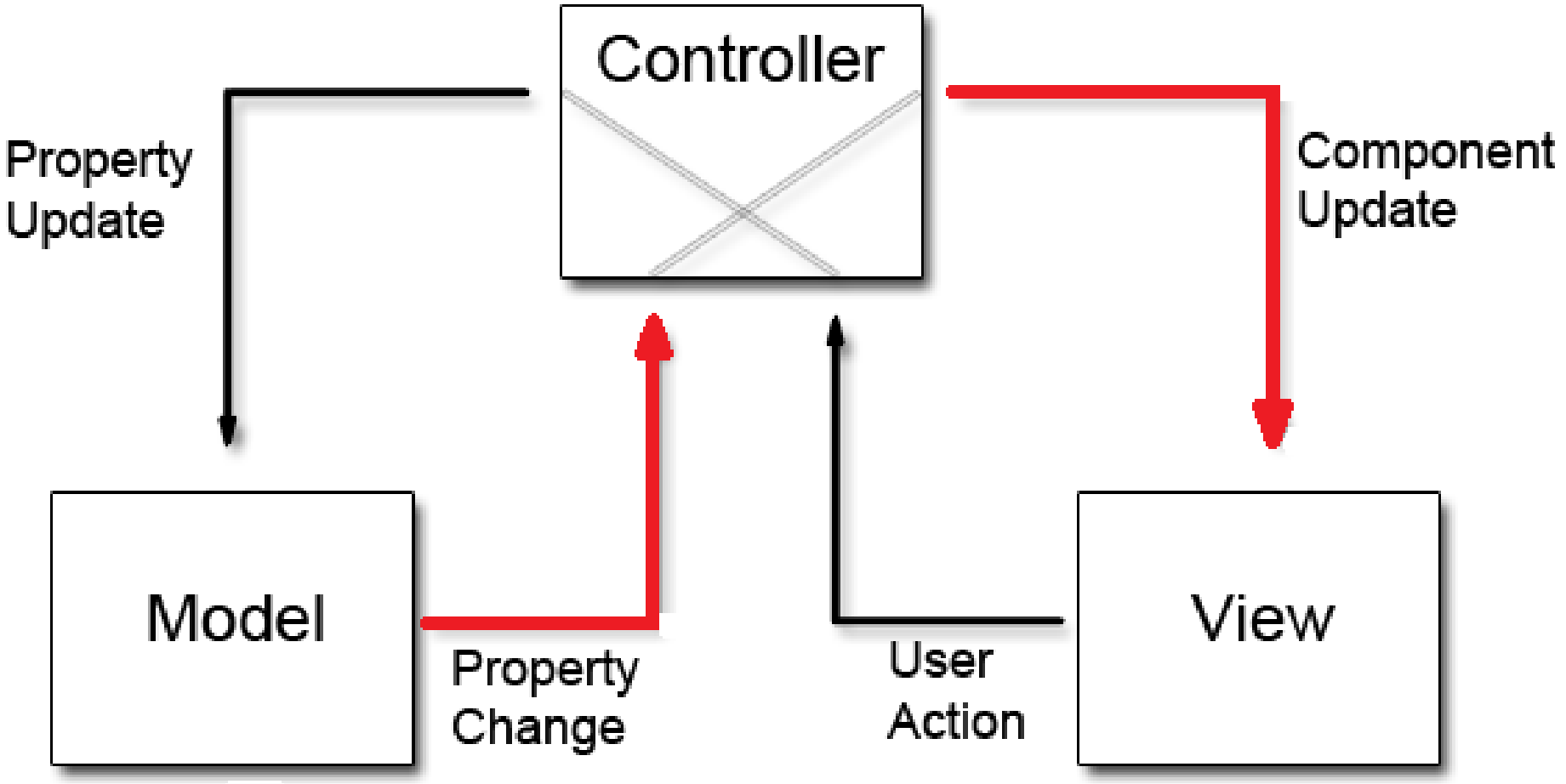
MVC – aktualizacja modelu

```
//gdzieś w kontrolerze
public void changeAge(String newValue){
   for (AbstractModel model: registeredModels) {
       model.setAge(newValue);
}

//w przykładzie 'do domu' użycie refleksji
//do szukania odpowiedniej metody w modelu
//(to jest rozwiązanie do celów szkoleniowych)
try{
 Method m = model.getClass().
 getMethod("set"+Prop, new Class[]{nV.getClass()});

 m.invoke(model, nV);
}
```

MVC – zdarzenia w modelu



MVC – zdarzenia w modelu

1. klasa java.beans.PropertyChangeSupport – wstawiamy jej instancję do klasy 'podmiotu' i delegujemy zadania (*np. add, notify*), umożliwia obsługę obserwatorów
2. metoda addPropertyChangeListener (*String propertyName*, PropertyChangeListener listener)
3. konstruktor PropertyChangeEvent (Object source, String property, Object old, Object new)
4. metoda firePropertyChange(PropertyChangeEvent evt) ≈ *notifyObservers()*
5. metoda propertyChange(PropertyChangeEvent evt) ≈ *update()*

MVC – zdarzenia w modelu

```
public abstract class AbstractModel{

    protected PropertyChangeSupport pCS =  new ...


    addPropertyChangeListener(PropertyChangeListener l)
    {
        pCS.addPropertyChangeListener(l);
    }


    firePropertyChange(PropertyChangeEvent evt)
    {
        pCS.firePropertyChange(evt);
    }
```

MVC – zdarzenia w modelu

```
public class Person extends AbstractModel {

// dodajemy kontroler do listy obserwatorów,
// będzie powiadomiony przez firePropertyChange

 String firstname;
 int age;

 setAge(int age) {
   int oldV = this.age;
   this.age = age;
   evt = new PropertyChangeEvent(this,“Age“,oldV,age);
   firePropertyChange(evt);
  }
```

MVC – kontroler pełni rolę mediatora

```
//kontroler  zdarzenie odebrane z modelu przesyła do widoków
public class Controller implements PropertyChangeListener
{
   private ArrayList<IView> registeredViews;
   public void propertyChange(PropertyChangeEvent evt) {

      for (IView view: registeredViews) {
         view.modelPropertyChange(evt);
      }
    }


//interfejs dla widoków
interface IView {
  public void modelPropertyChange(PropertyChangeEvent evt);
}
```

MVC – zdarzenie w modelu, zmiany w widoku

```
public class PersonView implements IView  {


modelPropertyChange(PropertyChangeEvent evt)
{

  if(evt.getPropertyName().equals("Age"))
  {
    String value = evt.getNewValue().toString();
    jTextFieldAge.setText(value);
  }
  else...
```

W3C

1. World Wide Web Consortium (W3C) – organizacja ustalająca standardy dla sieci WWW
2. W3C powstało w 1994, założycielem jest twórca WWW i hipertekstu Tim Berners Lee
3. Członkami są największe firmy informatyczne i uczelnie (np. Google, Microsoft, Apple, MIT)
4. Standardy ustalone przez W3C to m.in. URL, HTTP, CSS, HTML, **XML**, SVG,...

e**X**tensible **M**arkup **L**anguage

Standard XML ustalony w 1998, głównym zastosowaniem jest

1. przechowywanie danych
2. i przesyłanie danych

w ustrukturalizowany sposób niezależny od platformy i w formacie czytelnym dla ludzi

(*Celem HTML jest tylko prezentacja danych*)

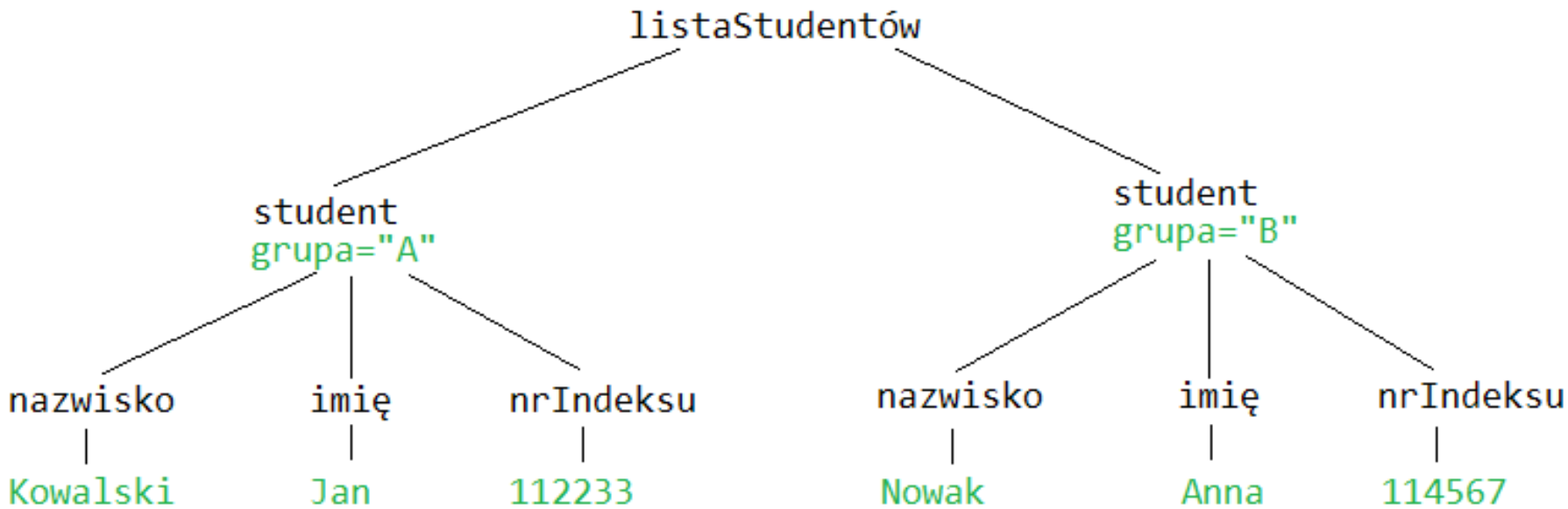
Zastosowania XML

- Maven (*Project Object Model już znacie*)
- Spring (*wstrzykiwanie zależności*)
- Hibernate (*mapowanie relacyjno-obiektowe*)
- Lekka trwałość, serializacja
- Komunikacja między "heterogenicznymi" aplikacjami
- Android (*np. opis interfejsu*)
- ...

```
<?xml version="1.0" encoding="utf-8"?>
<listaStudentow>
  <student grupa="A">
     <nazwisko>Kowalski</nazwisko>
     <imie>Jan</imie>
     <nrIndeksu>112233</nrIndeksu>
  </student>
  <student grupa="B">
     <nazwisko>Nowak</nazwisko>
     <imie>Anna</imie>
     <nrIndeksu>114567</nrIndeksu>
  </student>
</listaStudentow>
```

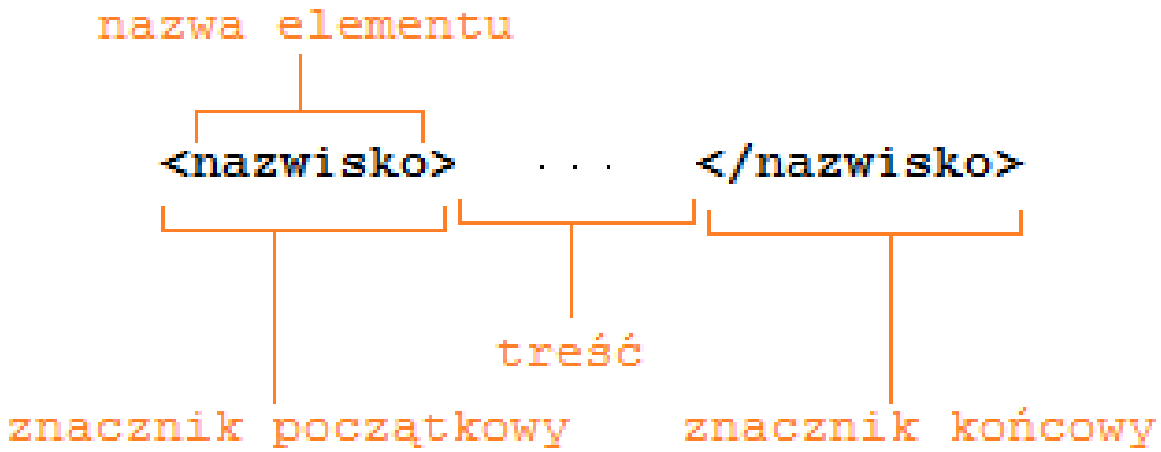
Struktura dokumentu XML – drzewo

1. Elementy zawierają inne – struktura drzewa. Korzeń drzewa (element główny) jest jeden
2. Elementy podrzędne (zawierane) stanowią lub zawierają informacje opisujące element nadrzędny



Elementy w XML – zasady

1. Pierwsza linia <? . . . ?> określa kodowanie
2. Każdy element ma dwa znaczniki i treśc
3. Brak z góry zdefiniowanych znaczników, nazwy mogą zawierać litery, cyfry, znaki – , . , _ i nie mogą się zaczynać od cyfr ani – , .



4. Elementy mogą zawierać inne, tekst, albo to i to:

   `<student> Pan <imie>Janek</imie> </student>`

5. Elementy bez treści można zapisać skrótowo:

   `<student></student>`

   jako

   `<student/>`

6. Przeplatanie elementów nie jest dozwolone:

   `<aa> <bb> </aa> </bb>`

7. Elementy (poza korzeniem) mogą się powtarzać i występować na różnych poziomach drzewa

8. Element może mieć atrybuty, nazwy atrybutów nie mogą się powtarzać w ramach jednego elementu

```
<student grupa=''A'' wzrost=''182''>...</student>
```

9. Kiedy używać atrybutów? Gdy dane mają prostą strukturę i niewielki zakres możliwych wartości

10. Komentarze: <!– komentarz –>

Przestrzenie nazw w XML

```
<publikacje>
    <artykul>
        <tytul> Answer to P=NP problem </tytul>
        <autorzy>Jan Kowalski</autorzy>
    </artykul>
</publikacje>
```

```
<pracownicy>
    <pracownik>
        <tytul> dr </tytul>
        <nazwisko>Kowalski</nazwisko>
    </pracownik>
</pracownicy>
```

Przestrzenie nazw w XML

Problem gdy chcemy połączyć dwa schematy: który tytuł mamy na myśli?

```
<artykul>
     <tytul> Answer to P=NP problem </tytul>
    <autorzy>
        <pracownik>
            <tytul> dr </tytul>
            <nazwisko>Kowalski</nazwisko>
        </pracownik>
    </autorzy>
</artykul>
```

Rozwiązanie: XML NameSpace (xmlns)

```
<pub:publikacje xmlns:pub="http://adres/xmlns/PUB"
                xmlns:prac="http://adres/xmlns/PRAC">
    <pub:artykul>
        <pub:tytul> Answer to P=NP problem </pub:tytul>
        <pub:autorzy>
            <prac:pracownik>
                <prac:tytul> dr </prac:tytul>
                <prac:nazwisko>Kowalski</prac:nazwisko>
            </prac:pracownik>
        </pub:autorzy>
    </pub:artykul>
</pub:publikacje>
```

Częste rozwiązanie — przestrzenie domyślne

```
<publikacje xmlns="http://adres/xmlns/PUB"
            xmlns:prac="http://adres/xmlns/PRAC">
    <artykul>
        <tytul>Dowody na ocieplenie klimatu</tytul>
        <autorzy>
            <prac:pracownik>
                <prac:tytul>dr</prac:tytul>
                ...
```

Np. pom.xml w Maven

```
<project xmlns="http://maven.apache.org/POM/4.0.0"... >
```

Walidacja dokumentów XML

1. XML poprawny składniowo (ang. well-formed) gdy jest zgodny z regułami składni
   - ma element główny (korzeń)
   - wszystkie elementy są poprawnie domknięte
   - wszystkie elementy są poprawnie zagnieżdżone
   - znaczniki są "case sensitive"
   - wartości atrybutów są w cudzysłowie
2. XML poprawny strukturalnie (ang. valid) gdy jest zgodny regułami definiującymi możliwe elementy i ich wzajemne powiązania
3. Do walidacji dokumentu służą specjalne języki, m.in. DTD oraz XSD (XML Schema Definition)

Document Type Definition (DTD)

DTD to standard umożliwiający zdefiniowanie struktury powiązan elementów

publikacja.xml

```
<!DOCTYPE publikacja SYSTEM "publikacja.dtd">
<publikacja>
  <tytul> ... </tytul>
  <autor> ... </autor>
</publikacja>
```

publikacja.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT publikacja (tytul, autor)>
<!ELEMENT tytul   (#PCDATA)>
<!ELEMENT autor   (#PCDATA)>
```

DTD składnia

1. Element ma parsowalną zawartość tekstową

```
<!ELEMENT tytul    (#PCDATA)>
```

2. Tylko jeden z elementów: tytul albo autor

```
<!ELEMENT publikacja (tytul | autor)>
```

3. Kolejno tytul i autor, autora można pominąć:

```
<!ELEMENT publikacja (tytul, autor?)>
```

DTD składnia

4. Dodatnia liczba tytułów, dowolna liczba autorów:

```
<!ELEMENT publikacja (tytul+, autor*)>
```

5. publikacja może mieć dowolną zawartość

```
<!ELEMENT publikacja ANY>
```

6. p może zawierać kombinacje tekstu i elementów

```
<!ELEMENT p (#PCDATA|a|b)*>
```

DTD składnia – definiowanie atrybutów

1. Dla każdego elementu w pliku XML można zdefiniować listę jego atrybutów ATTLIST (*tutaj element "pracownik"*)

```
<!ATTLIST pracownik   numer      ID              #REQUIRED
                      nazwisko   NMTOKEN         #REQUIRED
                      tytul      (mgr|dr|prof)   #IMPLIED>
```

2. Atrybut ma 3 składowe: nazwa, typ i informacja o wartości

3. Informacja o wartości atrybutu:
   - #IMPLIED - atrybut opcjonalny
   - #REQUIRED - atrybut obowiązkowy

DTD składnia – typy atrybutów

- PCDATA – parsowalny (przez parser XML) łańcuch znaków
  (*np. parser rozpoznaje elementy <> </>*)
- CDATA – nieparsowalny łańcuch znaków
  (*parser nie wnika w to co jest, nie rozpoznaje elementów*)
- NMTOKEN – pojedyncze słowo (litery, cyfry, znaki) ale bez znaków białych czyli spacji i tabulatorów
- ID – unikalny na cały dokument identyfikator elementu
- IDREF – referencja do jakiegoś ID w dokumencie, zawiera wartość ID stojącego w innym miejscu
- (x|y|z) – typ wyliczeniowy, jedna z wartości x, y lub z
- ...

XML Schema Definition (.xsd)

XSD - inny standard do opisu struktury XML

Różnice między DTD a XSD:

1. XSD oparte na składni XML, DTD ma własną
2. Składnia DTD jest bardziej zwięzła, czytelna (*zobacz przykład XSD dla pliku z publikacjami*)
3. XSD umożliwia bardziej elastyczne określanie typów oraz liczby wystąpień (*w DTD *, + oraz ?*)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="publikacja">
        <xs:complexType>
            <xs:sequence minOccurs="1" maxOccurs="unbounded">
                <xs:element ref="tytul" />
                <xs:element ref="autor" />
                <xs:element name="data" type="xs:date"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="tytul">
        <xs:complexType mixed="true" />
    </xs:element>

    <xs:element name="autor">
        <xs:complexType mixed="true" />
    </xs:element>

</xs:schema>
```

Zadanie domowe

Zwaliduj przykładowy plik na stronie:
https://www.xmlvalidation.com/