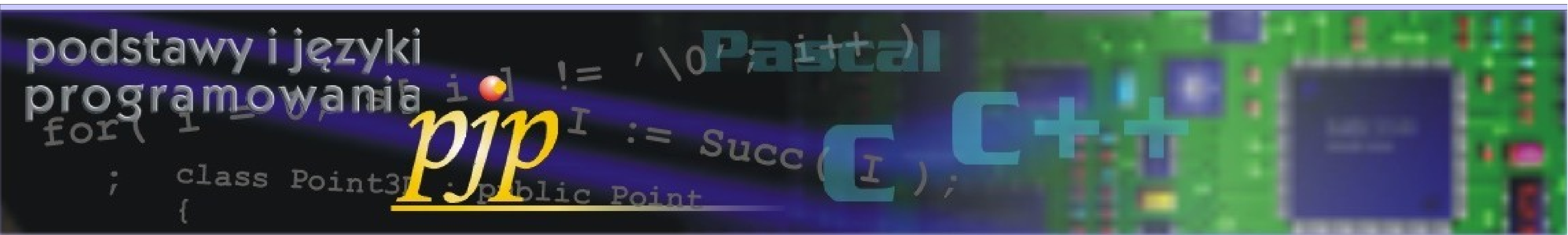
Projektowanie i programowanie obiektowe

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

Wzorce projektowe

Od delegowania do Strategii



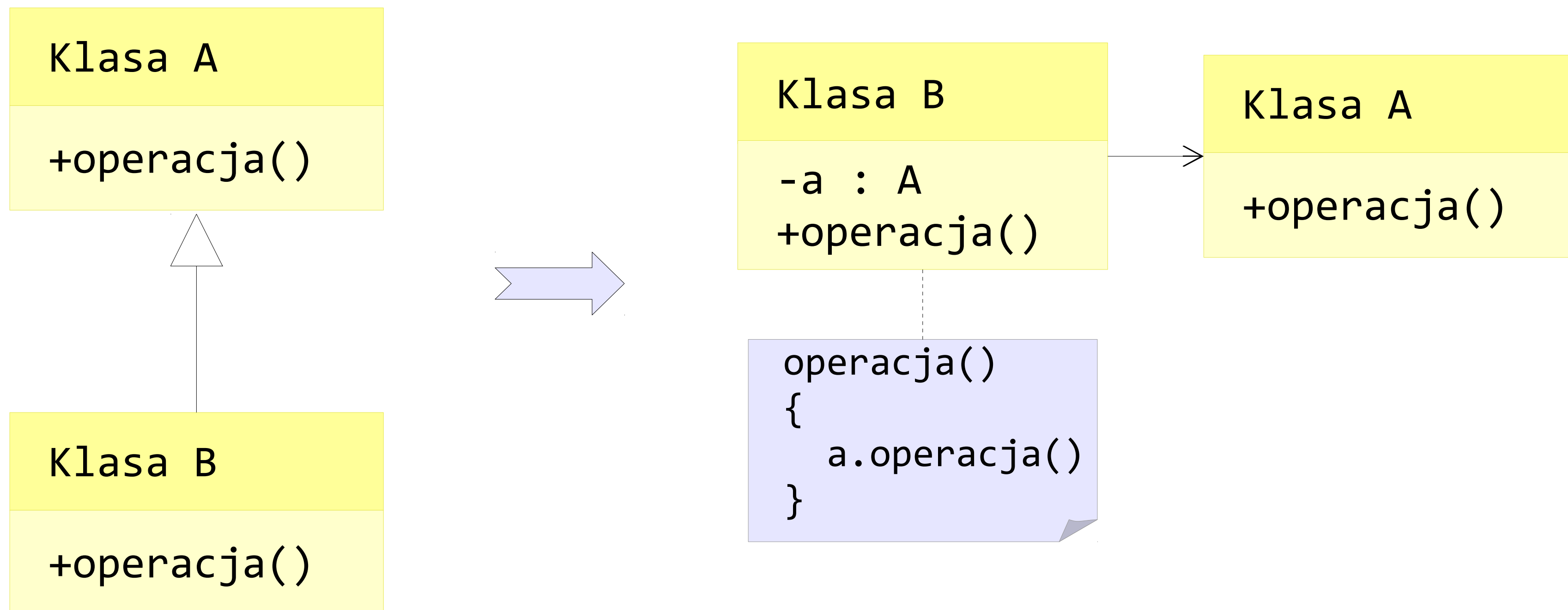
Koncepcja delegacji
Delegation

Delegacja – koncepcja ogólna

- Rozważając ***delegację*** na wysokim poziomie abstrakcji, zakłada ona, że wykonanie przez *obiekt* pewnej *operacji* może zostać przekazane do realizacji innemu *obiektowi*.
- Obiekt zlecający wykonanie operacji nazywamy *delegującym*, obiekt otrzymujący zlecenie nazywamy *delegowanym*.
- *Delegacja* nie wymaga występowania w języku programowania mechanizmu *delegatów* (np. C#).
- Delegowanie czasem bywa utożsamiane z *kompozycją obiektową*.

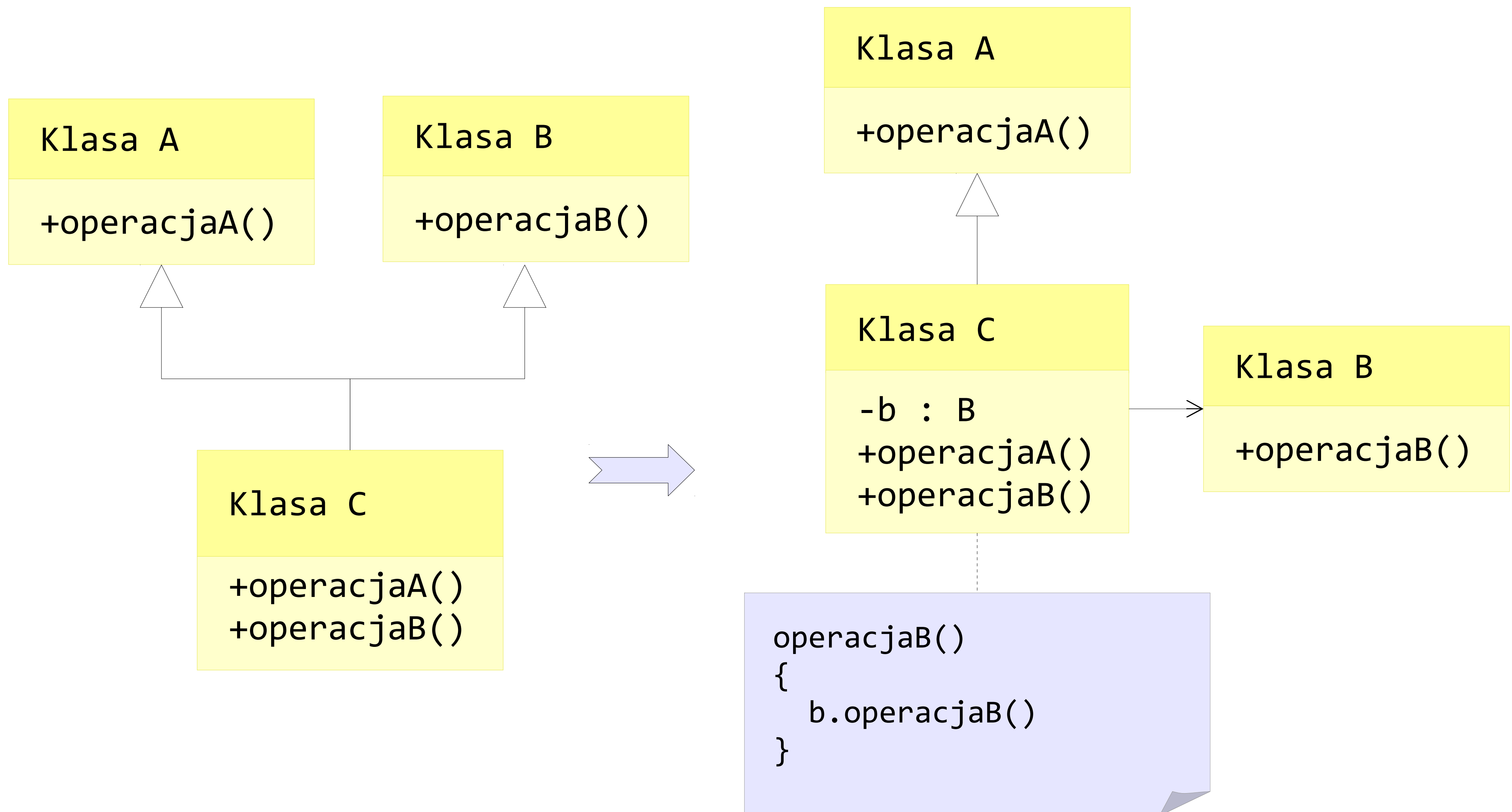
Delegacja - konkretniej

- ***Delegacja*** ma dostarczyć podejścia, które pozwoli na *unikanie stosowania dziedziczenia* w sytuacjach, w których jest ono niewygodne.



Delegacja - konkretniej

- Wzorzec ***Delegacja*** pozwoli osiągnąć podobne rezultaty do dziedziczenia wielobazowego w językach, które tego mechanizmu nie posiadają.



Delegacja - trywialny przykład

- Załóżmy, że człowiek, wykonuje swoją pracę poprzez zlecenie jej robotowi.

```
doWork()
{
  worker.doWork()
}
```

Human
-worker : Robot +doWork()

Robot
+doWork()

```
class Robot
{
    public void doWork()
    {
        System.out.println( "Robot: working..." );
    }
}

class Human
{
    Robot worker = new Robot();
    public void doWork()
    {
        worker.doWork();
    }
}

public class Delegacja01
{
    public static void main(String[] args)
    {
        Human john = new Human();
        john.doWork();
    }
}
```

Delegacja - trywialny przykład

- Załóżmy, że pracownik-człowiek, wykonuje swoją pracę poprzez zlecenie jej robotowi.

```
doWork()
{
  worker.doWork()
```

```
class Robot
{
    public void doWork()
    {
        System.out.println( "Robot: working..." );
    }
```

W tym przykładzie klasy *Robot* i *Human* są powiązane *delegacją*, osiągamy analogiczny efekt jak przy dziedziczeniu. Dziedziczenie nie byłoby tutaj sensowne (brak *relacji is a kind of*).

Jednak podobnie jak w przypadku dziedziczenia powiązanie pomiędzy klasami jest statyczne.

Co w sytuacji gdy roboty się zepsują...?

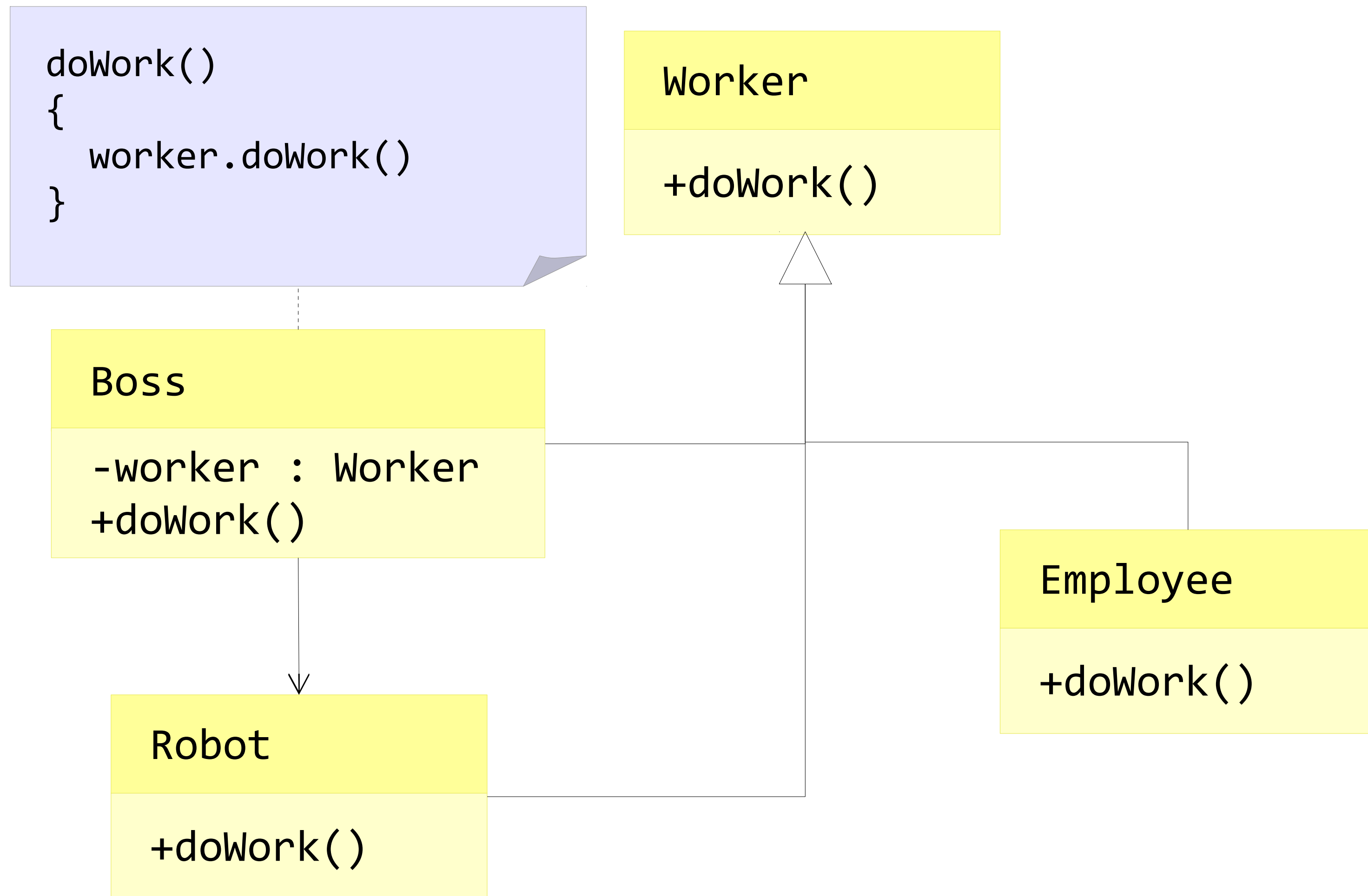
```
public class Delegacja01
{
    public static void main(String[] args)
    {
        Human john = new Human();
        john.doWork();
    }
}
```

Robot

+doWork()

Delegacja, dodajmy trochę abstrakcji, wspólny wzorzec

- Załóżmy, że pracować może człowiek lub robot. Pracownik-szef może zlecać wykonanie pracy robotowi albo innemu człowiekowi.



Delegacja - trochę lepszy przykład

```java
abstract class Worker
{
    public abstract void doWork();
}

class Robot extends Worker
{
    @Override
    public void doWork()
    {
        System.out.println( "Robot: working..." );
    }
}

class Employee extends Worker
{
    @Override
    public void doWork()
    {
        System.out.println( "Employee: working..." );
    }
}
```

Delegacja - trochę lepszy przykład

```
class Boss extends Worker
{
    Boss( Worker w )
    {
        worker = w;
    }

    @Override
    public void doWork()
    {
       System.out.print( "Boss working: " );
       worker.doWork();
    }

    private Worker worker;
}

. . .
Boss john = new Boss( new Robot() );
john.doWork();

Boss johnBoss = new Boss( new Employee() );
johnBoss.doWork();
```

```
Boss working: Robot: working...
Boss working: Employee: working...
```

Delegacja - trochę lepszy przykład

```
class Boss extends Worker
{
    Boss( Worker w )
    {
        worker = w;
    }

    @Override
    public void doWork()
    {
       System.out.print( "Boss working: " );
       worker.doWork();
    }

    private Worker worker;
}

. . .
Boss john = new Boss( new Robot() );
john.doWork();

Boss johnBoss = new Boss( new Employee() );
johnBoss.doWork();
```

„Wstrzyknięcie” obiektu delegowanego

```
Boss working: Robot: working...
Boss working: Employee: working...
```

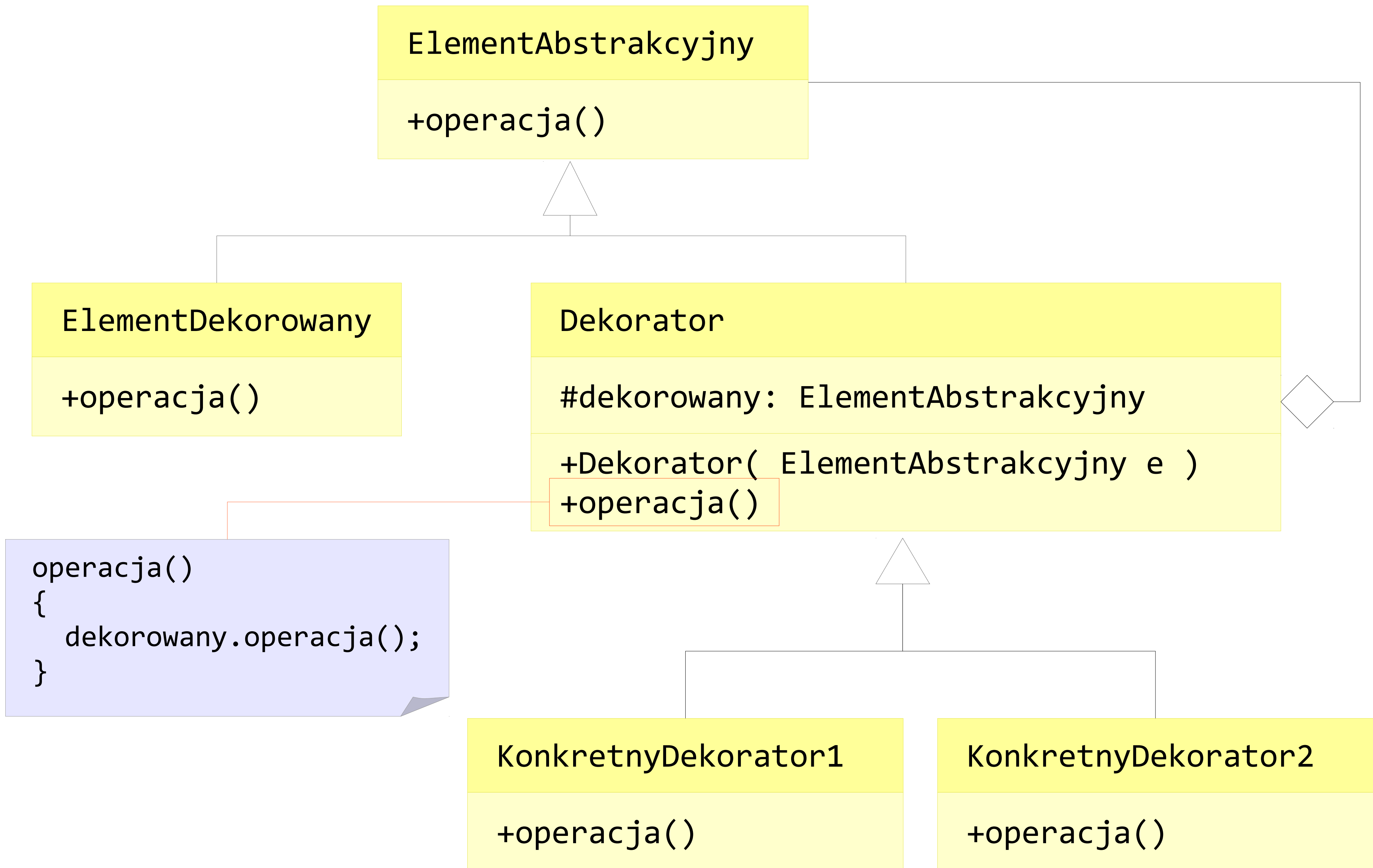
Wielokrotne delegowanie...

```
Boss john = new Boss( new Robot() );
Boss johnBoss = new Boss( john );
Boss CEO = new Boss( johnBoss );
CEO.doWork();
```

```
Boss working: Boss working: Boss working: Robot: working...
```

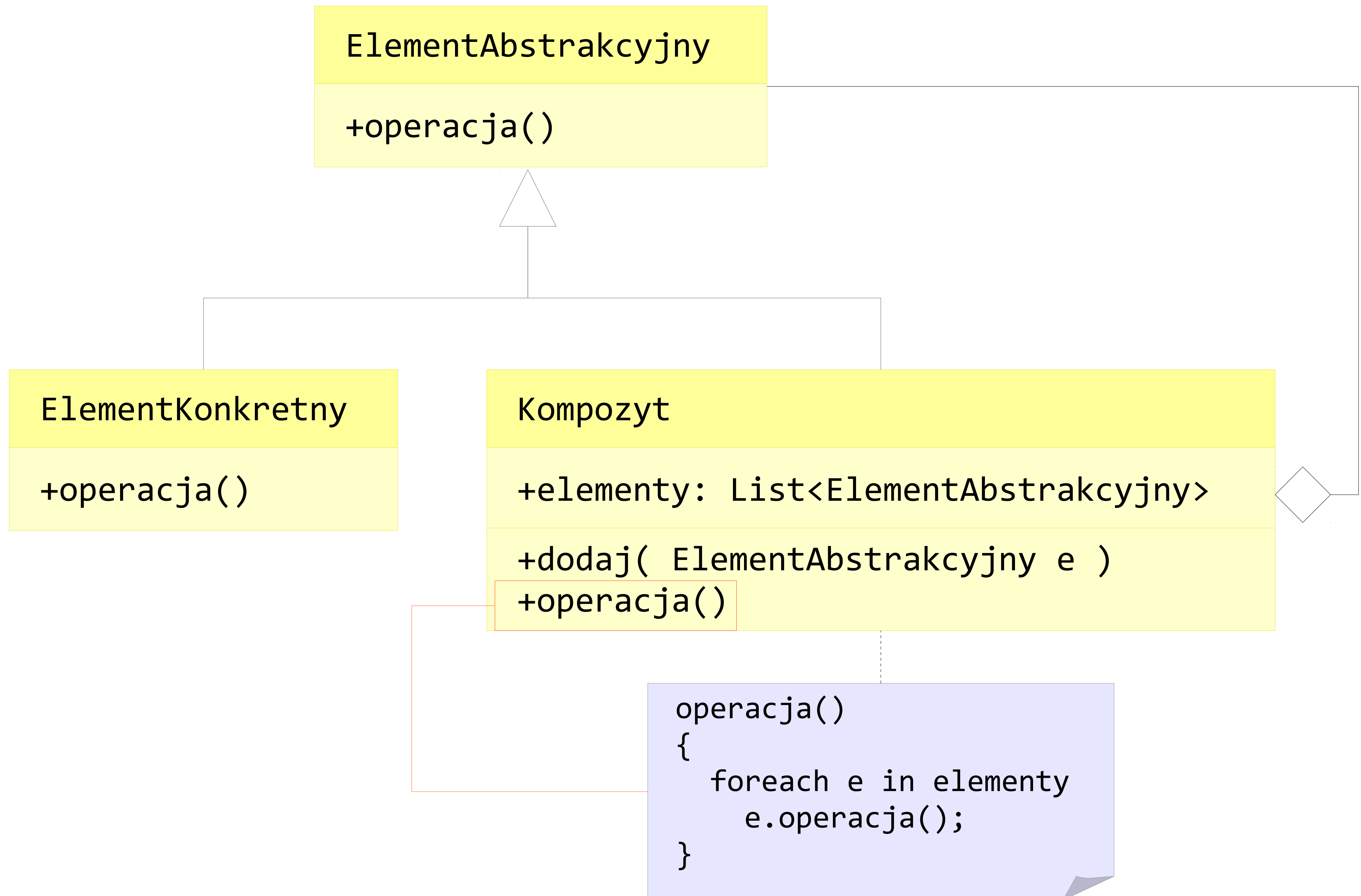
Czy to nam czegoś nie przypomina...?

Dekorator, schemat UML



A może coś jeszcze przypomina....

Kompozyt, schemat UML



Czym jest zatem delegowanie?

- Czy *delegowanie*, *delegacja* jest zatem wzorcem projektowym?
- A może jest pewną elastyczną koncepcją organizacji klas, równoległą a czasami alternatywną do dziedziczenia?
- *Delegowanie* pozwala na dynamiczne modyfikowanie powiązań pomiędzy obiektami oraz modyfikowanie zachowań obiektów.
- Stanowi (wraz z *abstrakcją* i *wstrzykiwaniem zależności*) ważny środek do budowania wzorców projektowych oraz budowania kodu podatnego na rozszerzenia a odpornego na modyfikacje.

Delegowanie...

Do czego można jeszcze wykorzystać delegowanie...

Strategia

- *Strategia* to operacyjny wzorzec projektowy umożliwiający zamienne stosowanie algorytmów reprezentowanych poprzez metody jednakowej definicji.
- Strategia definiuje wspólny interfejs dla każdego wykorzystywanego algorytmu.
- Należy zdefiniować metody realizujące konkretne algorytmy w klasach implementujących wspólny interfejs.
- Stosujemy wtedy, gdy wiele klas różni się tylko zachowaniem.
- Ale nie stosujemy dziedziczenia a kompozycję i delegację.

Klasa „kompresora plików” — różne algorytmy kompresji

```
KompresorPlików

-rodzajKompresji : int
-kompresujZIP( nazwaPliku: String );
-kompresujMP3( nazwaPliku: String );
+ustawRodzajKompresji( int rodzajKompresji )
+kompresujPlik( nazwaPliku: String )
```

```
kompresujPlik( nazwaPliku: String )
{
  switch( rodzajKompresji )
  {
    case ZIP: kompresujZIP( nazwaPliki );
              break;
    case MP3: kompresujMP3( nazwaPliki );
              break;
  }
}
```

Klasa „kompresora plików” — różne algorytmy kompresji

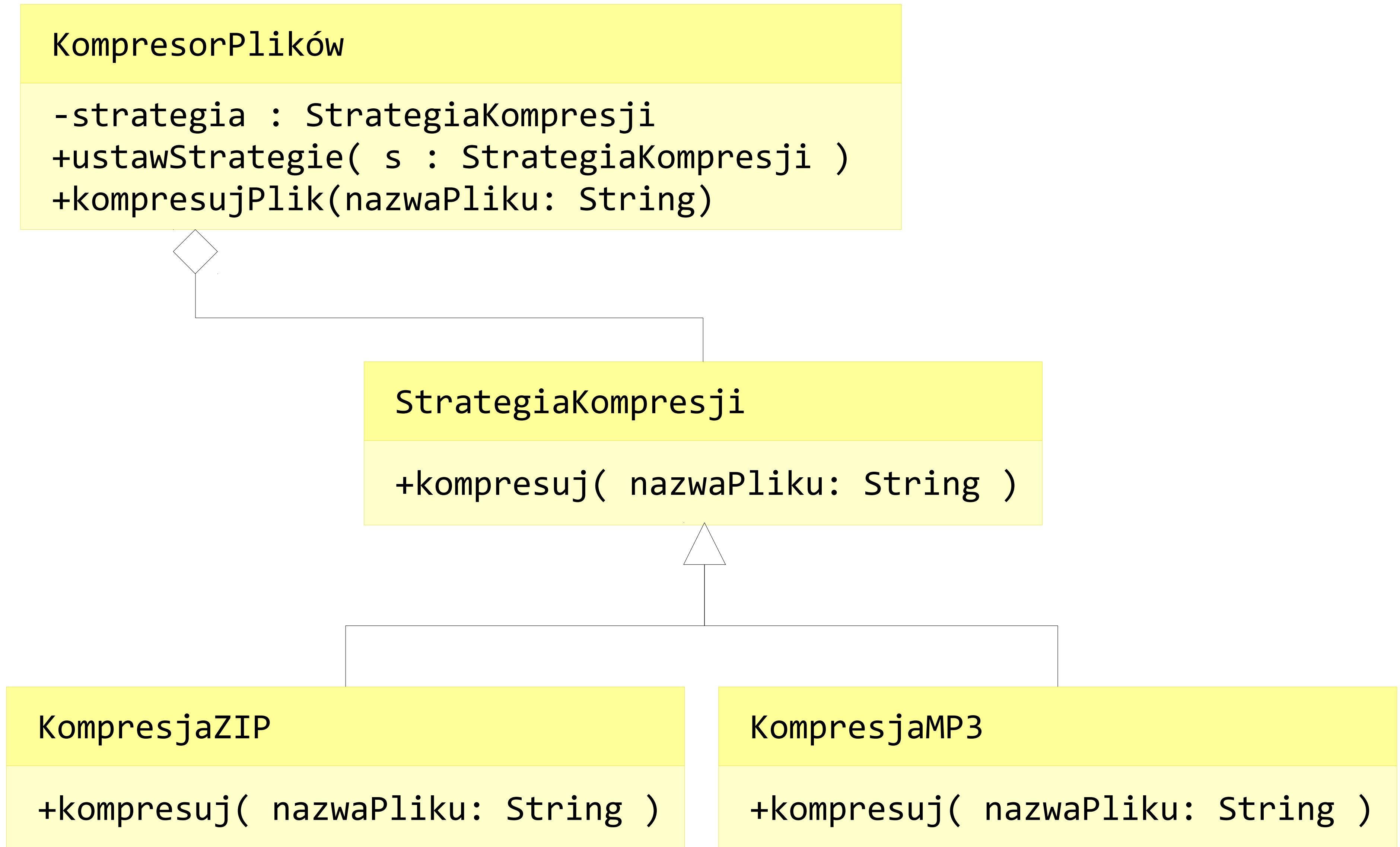
```
KompresorPlików

-rodzajKompresji : int
-kompresujZIP( nazwaPliku: String );
-kompresujMP3( nazwaPliku: String );
+ustawRodzajKompresji( int rodzajKompresji )
+kompresujPlik( nazwaPliku: String )
```

```
kompresujPlik( nazwaPliku: String )
{
  switch( rodzajKompresji )
  {
    case ZIP: kompresujZIP( nazwaPliki );
              break;
    case MP3: kompresujMP3( nazwaPliki );
              break;
  }
}
```

Ten kod nie spełnia zasady Open/Close...

Klasa „kompresora plików” — różne strategie kompresji



Przykład - różne metody kompresji plików

```java
interface CompressionStrategy
{
    public void compress( String fileName );
}

class ZIPCompressionStrategy implements CompressionStrategy
{
    @Override
    public void compress( String fileName )
    {
      System.out.println( "File: " + fileName + " compressed as ZIP" );
    }
}

class MP3CompressionStrategy implements CompressionStrategy
{
    @Override
    public void compress( String fileName )
    {
      System.out.println( "File: " + fileName + " compressed as MP3" );
    }
}
```

Przykład - różne metody kompresji plików

```
class FileCompressor
{
    public void compressFile( String fileName )
    {
      strategy.compress(fileName);
    }
    public void setCompression( CompressionStrategy s )
    {
        strategy = s;
    }
    private CompressionStrategy strategy;
}

. . .
FileCompressor comp = new FileCompressor();

comp.setCommpression( new ZIPCompressionStrategy() );
comp.compressFile( "program.java" );

comp.setCommpression( new MP3CompressionStrategy() );
comp.compressFile( "song.wav" );
```

```
File: program.java compressed as ZIP
File: song.wav compressed as MP3
```

Przykład - różne metody kompresji plików

```
class FileCompressor
{
    public void compressFile( String fileName )
    {
      strategy.compress(fileName);
    }
    public void setCompression( CompressionStrategy s )
    {
        strategy = s;
    }
    private CompressionStrategy strategy;
}

. . .
FileCompressor comp = new FileCompressor();

comp.setCommpression( new ZIPCompressionStrategy() );
comp.compressFile( "program.java" );

comp.setCommpression( new MP3CompressionStrategy() );
comp.compressFile( "song.wav" );
```

„Wstrzyknięcie” obiektu strategii
Zachowujemy zasadę Open/Close

```
File: program.java compressed as ZIP
File: song.wav compressed as MP3
```

Strategia ustalana wewnętrznie przez obiekt kontekstu wzorca

```
class FileCompressor
{
    . . .

    public void autoCompressFile( String fileName )
    {
       if( fileName.endsWith( ".java" ) )
         ( strategy = new ZIPCompressionStrategy() ).compress(fileName);
       if( fileName.endsWith( ".wav" ) )
         ( strategy = new MP3CompressionStrategy() ).compress(fileName);
    }

    private CompressionStrategy strategy;
}
```

```
comp.autoCompressFile( "program.java" );
comp.autoCompressFile( "song.wav" );
```

```
File: program.java compressed as ZIP
File: song.wav compressed as MP3
```

Ten kod nie spełnia zasady Open/Close... ale nie dajmy się zwariować i stosujmy KISS...

Strategia - schemat ogólny