bns it

Moduł 7

WZORCE KREACYJNE GOF

Niniejszy plik jest materiałem reklamowym BNS IT s.c. i pozostaje własnością intelektualną BNS IT s.c.. Może być rozpowszechniany tylko w takiej postaci w jakiej jest. Używanie zawartych tu treści i form bez zaznaczenia autorstwa i pochodzenia pliku, zwłaszcza na użytek prowadzenia szkoleń, wykładów i wystąpień publicznych, jest zabronione.

bns it

Czego się nauczysz?

- Jak implementować wzorce:
  - *Simple Factory**
  - *Factory Method*
  - *Abstract Factory*
- Kiedy używać powyższych wzorców
- Jaka jest relacja pomiędzy UML a kodem Java

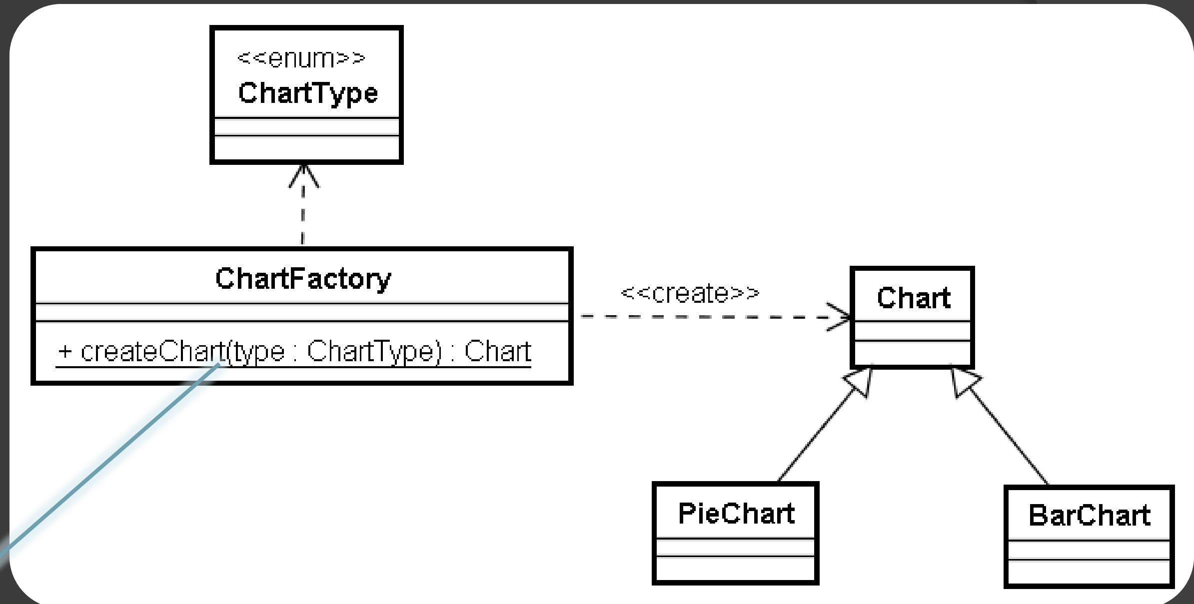
* Niektórzy nie pozwalają nazywać *Simple Factory* wzorcem projektowym (...) Więcej o niuansach wzorców projektowych w szkoleniu **Wzorce projektowe i refaktoryzacja do wzorćów.**

bns it

Simple Factory

Metoda *Simple Factory* decyduje o utworzeniu obiektu na podstawie parametrów wejściowych.

Bardzo często jest statyczna.




```
public class ChartFactory {

  public static Chart createChart( ChartType type
) {

    if ( ChartType.BAR.equals( type ) ) {
      return new BarChart();
    } else if ( ChartType.PIE.equals( type ) ) {
      return new PieChart();
        //...
```

bns it

Simple Factory(2)

- Używaj *Simple Factory* jako nazwanego zamiennika konstruktora

Użycie wersji statycznej

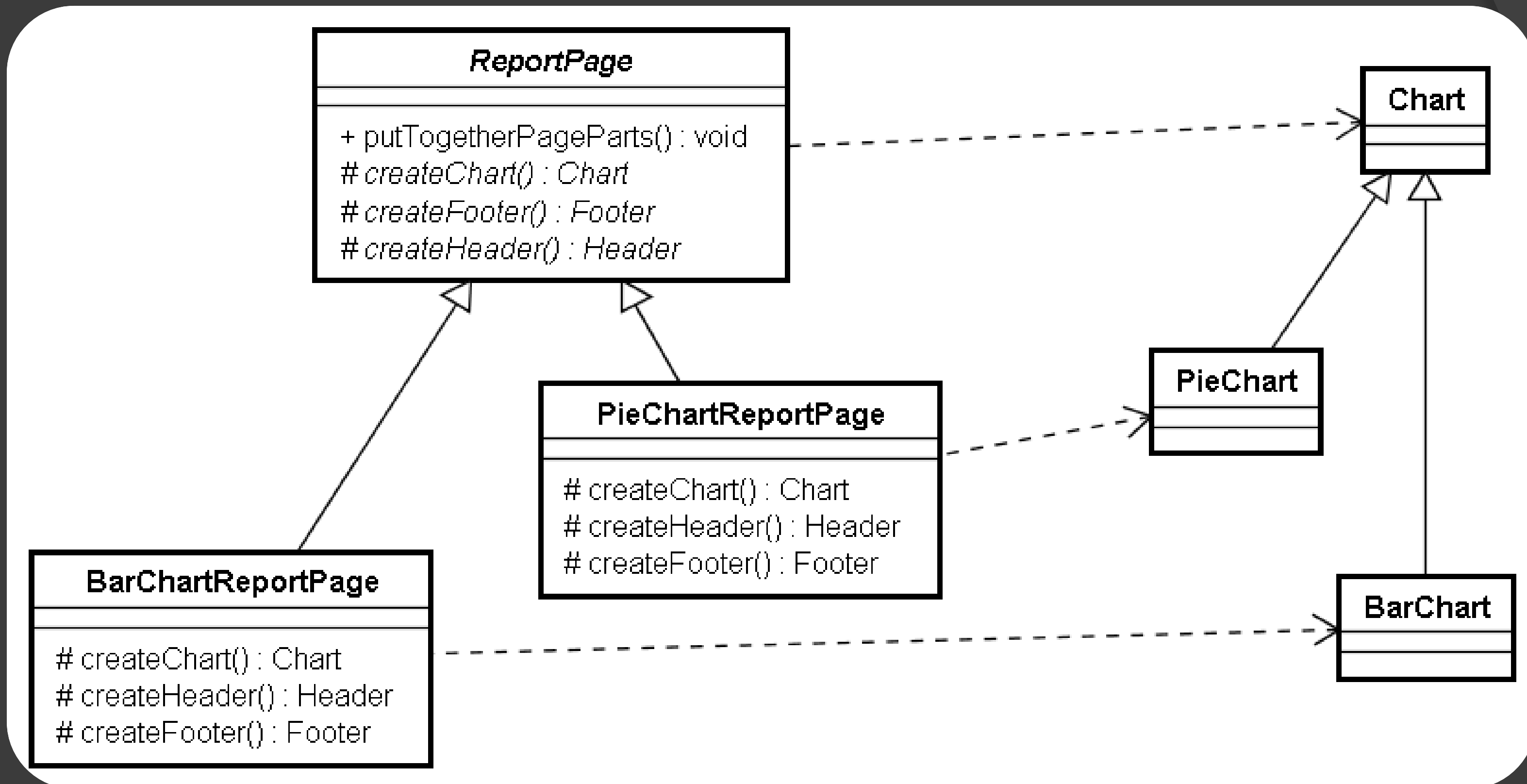
```
public class ChartManager {

  public void drawChart() {

    Chart chart = ChartFactory.createChart( ChartType.BAR );
    //...
```

Użycie wersji niestatycznej

```
public class ChartManager {

  private ChartFactory chartFactory;

  public void drawChart() {

    Chart chart = chartFactory.createChart( ChartType.BAR );
    //...
```

bns it

Factory Method



- Klasa *ReportPage* definiuje metodę tworzącą wykres (metodę fabrykującą, *Factory Method*) i posługuje się nią w swoich algorytmach, lecz implementację pozostawia klasom pochodnym

bns it

Factory Method(2)

```
public abstract class ReportPage {

  protected abstract Chart createChart();

  public void putTogetherPageParts() {
    //...

    Chart chart = createChart();

    //...
  }
}
```

Metoda szablonowa (*Template Method*) *putToghetherPageParts()* definiuje ogólny algorytm tworzenia strony raportu. Wykorzystywana jest tam metoda fabrykująca.

O typie konkretnego wykresu decydują klasy pochodne np. *PieChartReportPage*, implementując metodę fabrykującą.

```
public class PieChartReportPage
extends ReportPage {

  @Override
  protected Chart createChart() {
    return new PieChart();
  }
}
```

bns it

Factory Method(3)

```
public class ReportManager {

  public void fillPage( ReportPage page ) {
    page.putTogetherPageParts();
  }
}
```

- Dzięki *Factory Method* możesz obsługiwać stronę raportu i nie martwić się jaki konkretnie wykres zostanie utworzony
- Używaj *Factory Method* w kontekście metody szablonowej (*Template Method*), w której określisz ogólny algorytm (tworzenie strony raportu), a szczegóły (tworzenie konkretnego wykresu) pozostawisz klasom pochodnym
- Zauważ że dodanie nowego wykresu wymaga tylko dodania nowej klasy! W istniejącym kodzie nic się nie zmienia!

bns it

A jeśli...

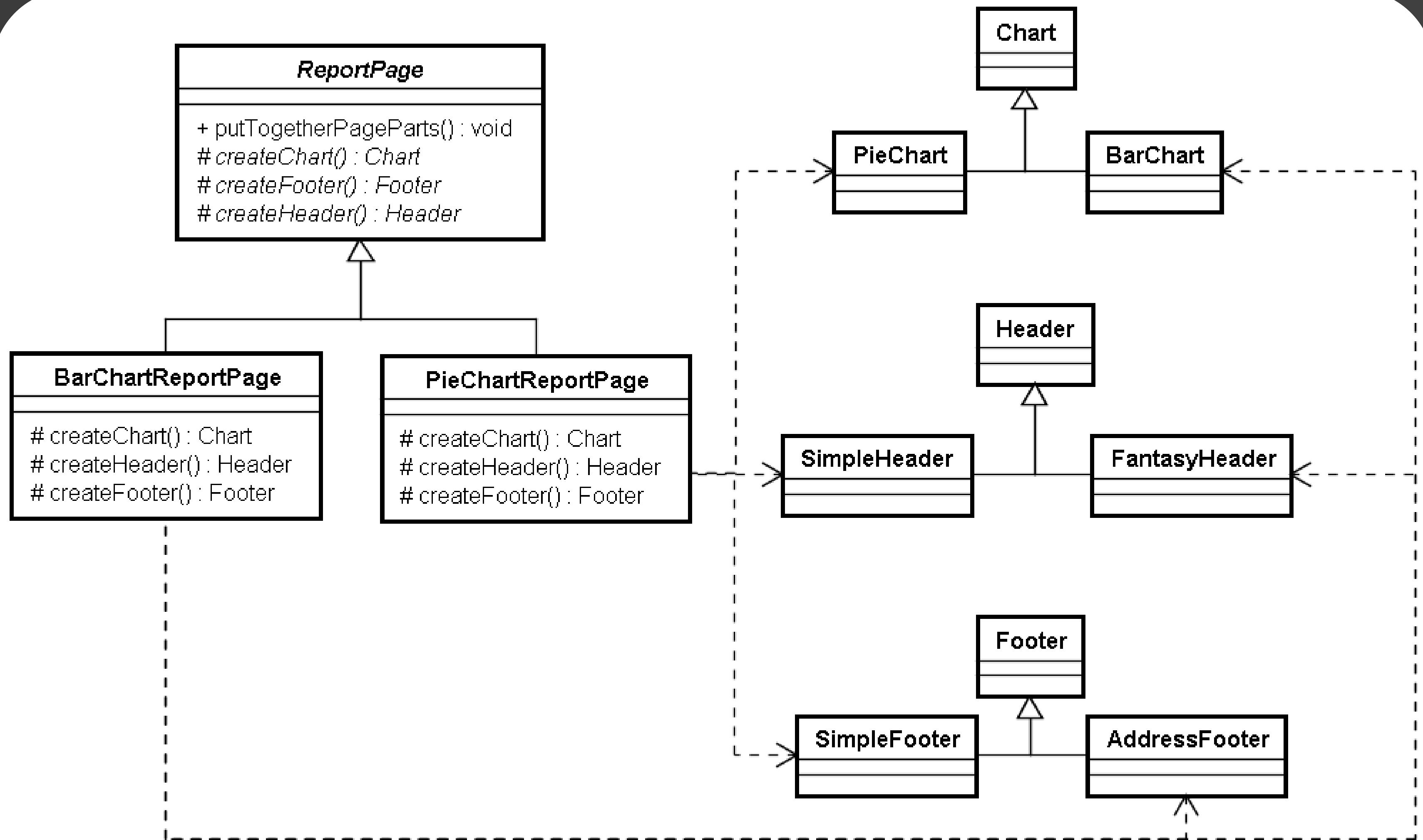
- strona raportu składa się dodatkowo z nagłówka, stopki, no i z wykresu
- strony z wykresem *pie* mają inne stopki i nagłówki niż strony z wykresem *bar*
- wciąż chcesz utrzymać niezależność używania strony od szczegółów implementacji

To w konsekwencji potrzebujesz

- zarządzania wytwarzaniem rodziny powiązanych produktów (różne wykresy, stopki, nagłówki)

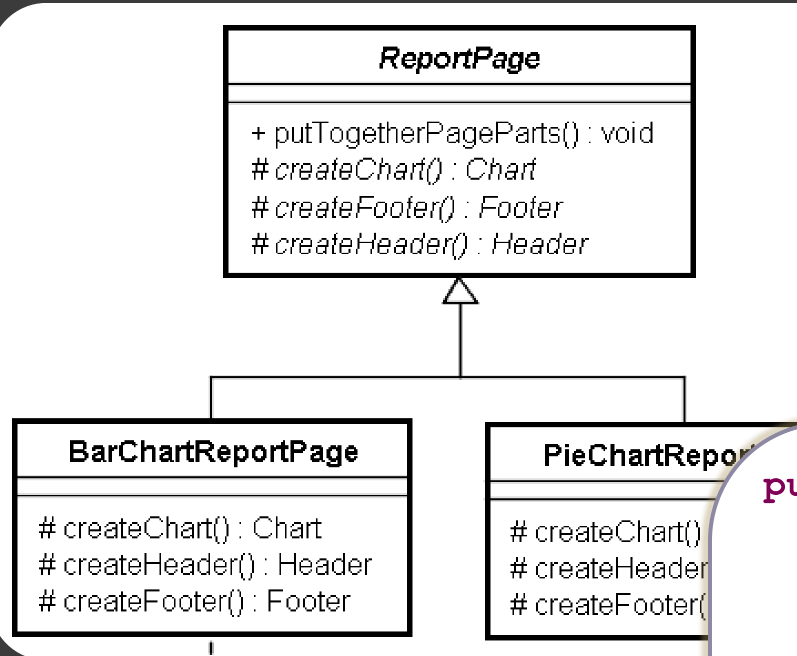
bns it

Abstract Factory



bns it

Abstract Factory(2)




```java
public abstract class ReportPage {

  protected abstract Chart createChart();
  protected abstract Header createHeader();
  protected abstract Footer createFooter();

  public void putTogetherPageParts() {
    //...
    Chart chart = createChart();
    Header header= createHeader();
    Footer footer = createFooter();
    //...
  }
}
```

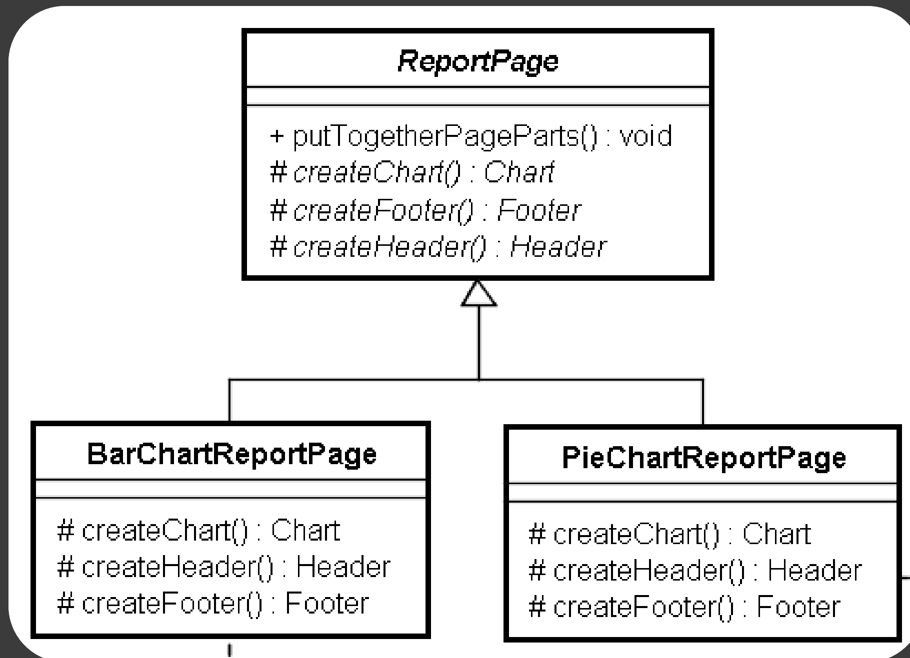
bns it

Abstract Factory(3)

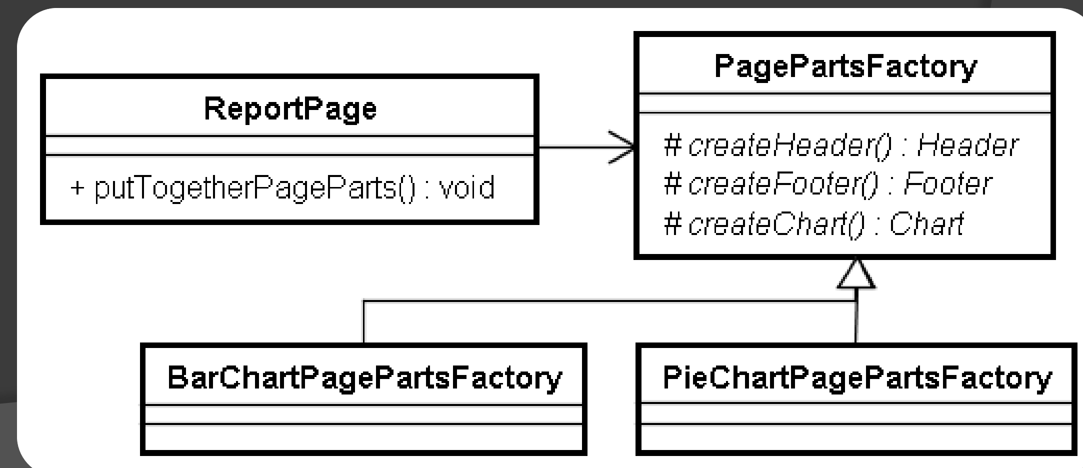
- Wzorzec *Abstract Factory* odpowiada za tworzenie **rodziny powiązanych ze sobą produktów**
- Cała złożoność implementacji jest ukryta przed klientem, który pracuje tylko z abstrakcyjną klasą *ReportPage*
- Zauważ, że w przykładzie *Abstract Factory* został zaimplementowany z użyciem *Factory Method*
- *Abstract Factory* podobny do *Factory Method* lecz nacisk kładzie na **rodzinę produktów**

bns it

Abstract Factory(4)

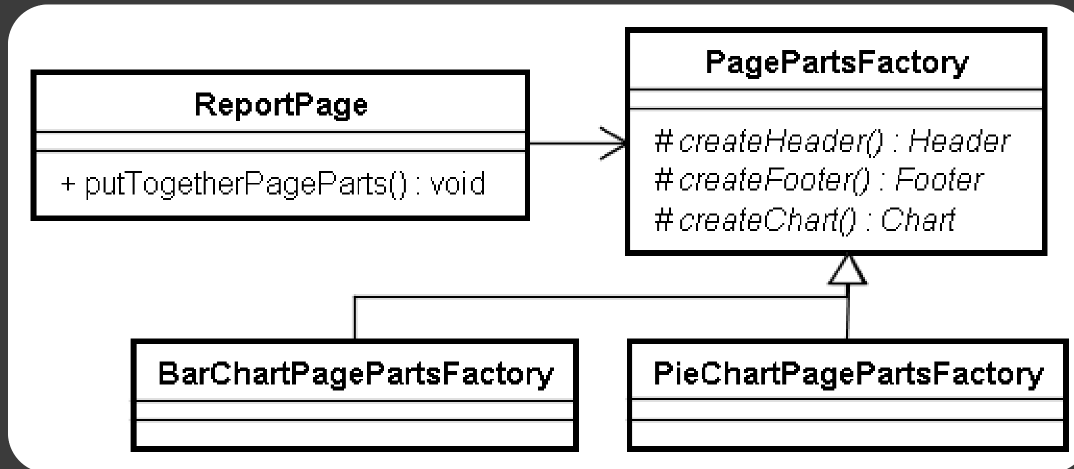


W miarę rozrastania się odpowiedzialności klasy *ReportPage*, warto wydzielić do osobnej klasy tworzenie składowych strony.



bns it

Abstract Factory(5)



Od tego momentu obiekty tworzące części strony można podmieniać **dynamicznie**, natomiast sama strona *ReportPage* pozostaje niezmieniona.

Efekt ten został uzyskany dzięki preferencji **kompozycji** nad **dziedziczenie**.

```
public class ReportPage {

  private PagePartsFactory pagePartsFactory;

  public void putTogetherPageParts() {

    Chart chart = pagePartsFactory.createChart();
    Header header = pagePartsFactory.createHeader();
    Footer footer = pagePartsFactory.createFooter();
    //...
  }
}
```

bns it

Kontakt z BNS IT

BNS IT

Al. Wyszyńskiego 22/17

94-042 Łódź

E-mail: bnsit@bnsit.pl

Tel.:+48 42 209 38 45

Fax.: +48 42 209 38 63