Programowanie obiektowe

Wykład 5: Interfejsy

dr inż. Marcin Luckner
mluckner@mini.pw.edu.pl

Wydział Matematyki i Nauk Informacyjnych

Wersja 1.3
9 marca 2022

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Abstrakcja

- Klasa tworzy szablon, który może być użyty do stworzenia obiektu.
- Jednakże w Javie występują także typy abstrakcyjne. Nie służą do tworzenia obiektów, ale do organizacji hierarchii klas i do przekazywania informacji o zachowaniu obiektów.
- Są to *klasy abstrakcyjne* i *interfejsy*.

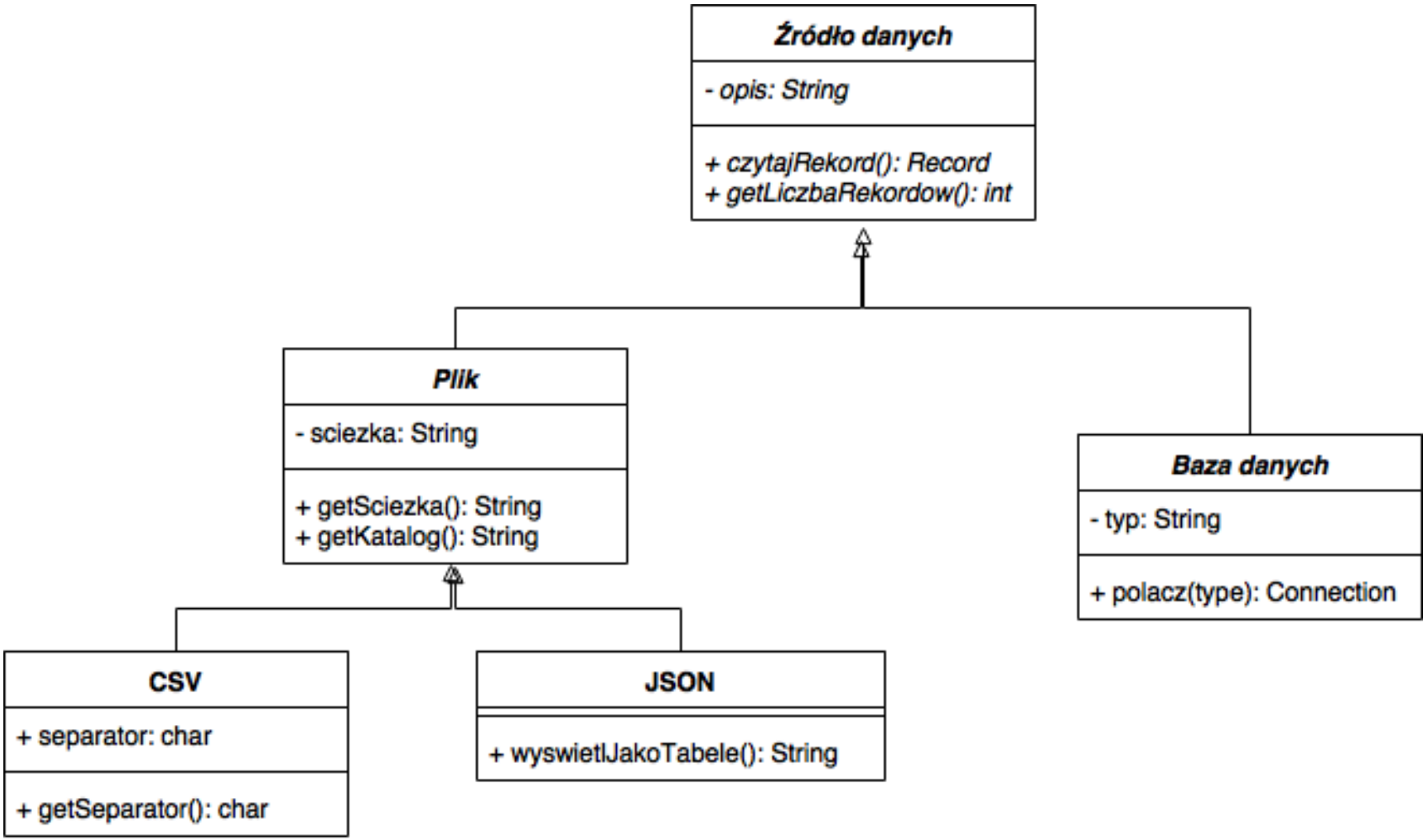
Klasy abstrakcyjne

- Klasy leżące wyżej w hierarchii klas mają bardziej ogólny charakter.
- Nie są już wzorcem dla obiektów, a stają się wzorcem dla innych klas.
- Są to zazwyczaj klasy abstrakcyjne, takie jak `Number`, które nie pozwalają na tworzenie swoich instancji.
- Wynika to z zadeklarowania wewnątrz nich metod abstrakcyjnych, których implementacja jest nieokreślona.
- Klasa abstrakcyjna, oprócz metod abstrakcyjnych, może zawierać zwyczajne metody i pola.

Metody abstrakcyjne

- Metody abstrakcyjne są oznaczone modyfikatorem `abstract`.
- Ich deklaracja w klasie ogranicza się do podania typu, nazwy i argumentów metody.
  - `abstract double doubleValue()` w klasie `Number`.
- Mogą być umieszczane tylko w klasach abstrakcyjnych, oznaczonych modyfikatorem `abstract`.
- Muszą być nadpisane w klasach pochodnych, lub klasa pochodna też musi być abstrakcyjna

Hierarchia z klasami abstrakcyjnymi



Rysunek 1: Klasy `ZrodloDanych`, `Plik` i `BazaDanych` są klasami abstrakcyjnymi

Przykład klasy abstrakcyjnej

Klasa Tendency określa tendencję szeregu czasowego

```
1  public abstract class Tendency {
2
3      double[] timeSeries;
4
5      public abstract double calculateTendency();
6
7      public int getSeriesLength() {
8          return timeSeries.length;
9      }
10
11     public Tendency(double[] timeSeries) {
12         this.timeSeries = timeSeries;
13     }
14 }
```

- Ponieważ tendencja może być liczona na różne sposoby, metoda `calculateTendency` pozostaje abstrakcyjna.
- Klasa zawiera także metody określone jak pole `timeSeries` i metoda `getSeriesLength`.
- Chociaż klasa nie umożliwia tworzenia instancji, możemy w niej zaimplementować konstruktor.

Przykłady klas pochodnych - moda

Klasa Mode[1]

```
1  public class Mode extends Tendency {
2      public Mode(double[] timeSeries) {
3          super(timeSeries);
4      }
5
6      @Override
7      public double calculateTendency() {
8          double modeValue, maxCount;
9          maxCount = 0;
10         modeValue = Double.NaN;
11         for (double value: timeSeries) {
12             int count = 0;
13             for (double occurrence: timeSeries) {
14                 if (value == occurrence) count++;
15             }
16             if (count > maxCount) {
17                 maxCount = count;
18                 modeValue = value;
19             }
20         }
21         return modeValue;
22     }
23 }
```

[1] Kod klasy nie jest optymalny wydajnościowo, poprawimy to w przyszłości

Przykłady klas pochodnych - średnia

Klasa Mean

```
1  public class Mean extends Tendency {
2      public Mean(double[] timeSeries) {
3          super(timeSeries);
4      }
5
6      @Override
7      public double calculateTendency() {
8          double meanValue;
9
10         meanValue = 0;
11         for (double value: timeSeries) {
12             meanValue+= value;
13         }
14         return meanValue/getSeriesLength();
15     }
16 }
```

Wykorzystanie klas abstrakcyjnych

- Możemy wykorzystać abstrakcję, by określić wstępne założenia, jak konieczność policzenia tendencji dla szeregu czasowego.

```
1  double[] timeSeries = {4.5, 4.0, 5.0, 4.0};
2  Tendency tendency;
```

- W zależności od potrzeb, stosujemy właściwą implementację

```
1  tendency = new Mean(timeSeries);
2  System.out.println("The average for " +
       tendency.getSeriesLength() +" marks is " +
       tendency.calculateTendency());
3
4  tendency = new Mode(timeSeries);
5  System.out.println("The most popular wage among " +
       tendency.getSeriesLength() +" bets is " +
       tendency.calculateTendency());
```

```
The average for 4 marks is 4.375
The most popular wage among 4 bets is 4.0
```

Interfejs

- Interfejs przedstawia wymagania wobec zachowania obiektu, który go implementuje.
- Dzięki interfejsowi można obsługiwać obiekty bez stawiania wymagań co do ich typu.
- Interfejs nie może mieć innych metod niż abstrakcyjne.
- Muszą być nadpisane w klasach implementujących interfejs.
- Jedna klasa może implementować wiele interfejsów.

Definicja interfejsu

- Przykładem definicji interfejsu jest `interface Comparable`.

```
1  public interface Comparable{
2    int compareTo(Object other);
3  }
```

- Interfejs możemy zaimplementować w naszej klasie

```
1  public class Employee extends Person implements
       Comparable {
2      private double salary;
3      @Override //Bezpieczna metoda wymaga programowania
           generycznego
4      public int compareTo(Object o) {
5          return (int)(salary-((Employee)o).salary); //
6      }
7  }
```

- Używając metody `sort(Object[] a)` klasy `Arrays` możemy teraz posortować pracowników względem wynagrodzenia, choć metoda nie jest dedykowana obsłudze klasy `Employee`.

Drukowanie bez interfejsu

- Zdefiniujmy klasę `Printer`, która ma drukować różne typy dokumentów

Klasa `Printer`

```
1  public class Printer {
2
3      public void print(TextDocument textDocument) {
4          //Print text document
5      }
6      public void print(Image image) {
7          //Print image
8      }
9  }
```

- Klasa musi posiadać metody do obsługi różnych typów dokumentów
- Chcąc wydrukować nowy typ dokumentum musimy zmodyfikować klasę `Printer`.

Określanie interfejsu

- Załóżmy, że drukowanie wymaga od nas dwóch rzeczy
  - przetworzenia dokumentu do postaci binarnej
  - określenia liczby stron wydruku
- Możemy założyć, że każdy dokument będzie posiadał metody realizujące te zadania

```
1       public byte[] print();
2       public int count(Printer printer);
```

- Potrzebna jest jeszcze gwarancja, że obiekty, które chcemy wydrukować będą posiadać te metody.

Drukowanie z interfejsem

- Zdefiniujmy interfejs `Printable`

Interfejs `Printable`

```
1  public interface Printable {
2      byte[] print();
3      int count(Printer printer);
4  }
```

- Przedefiniujmy klasę `Printer`

Klasa `Printer`

```
1  public class Printer {
2      public void print(Printable document) {
3          //Print document
4      }
5  }
```

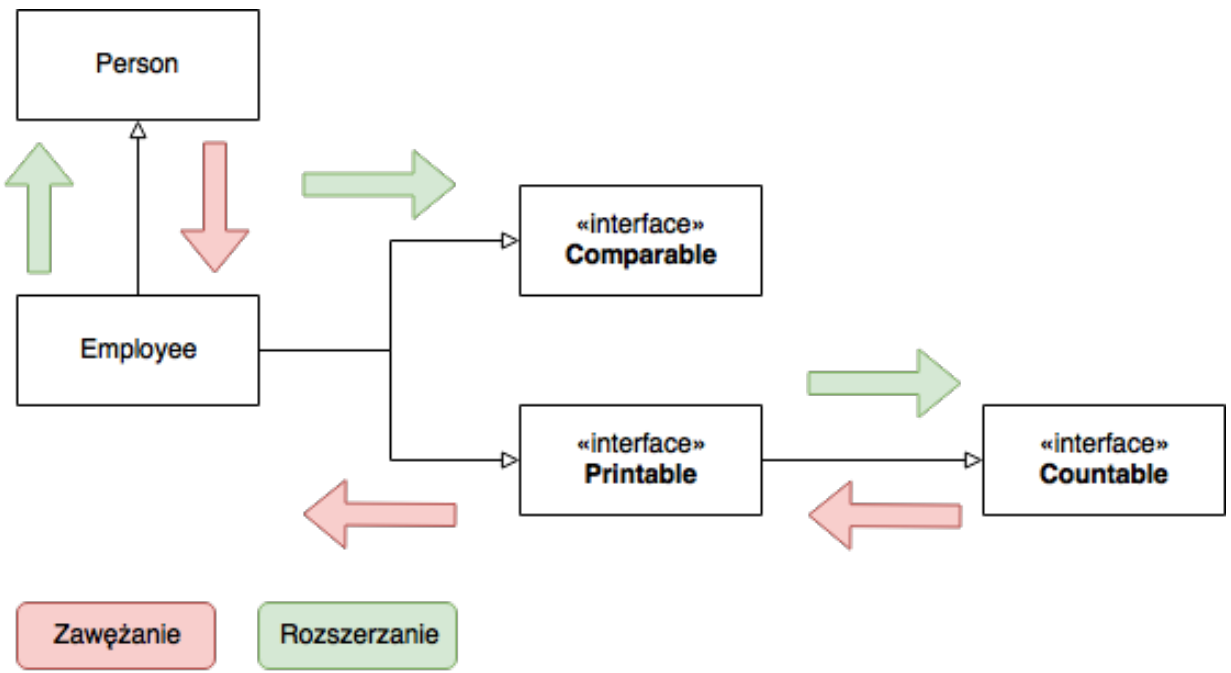
- Twórca dokumentów musi zapewnić implementację odpowiednich metod w dokumentach, ale klasa `Printer` może wydrukować każdy dokument implementujący `Printable`.

Właściwości interfejsów

- Wszystkie składowe są publiczne.
- Wszystkie metody są abstrakcyjne.
- Wszystkie pola są statyczne, finalne i publiczne.
- Dlatego modyfikatory są zbędne.
- Za to każde pole musi być zainicjowane.

Interfejsy w hierarchii klas

- Jak w przypadku klas abstrakcyjnych, dla interfejsu nie można utworzyć instancji, ale można użyć go jako referencji.
- Dopuszcza się konwersje rozszerzającą i zawężającą, ale tylko rozszerzająca jest bezpieczna.



Rysunek 2: Hierarchia klas z uwzględnieniem interfejsów

Interfejsy, a klasy abstrakcyjne

- Klasa abstrakcyjna
  - Służy do opisu klas potomnych.
  - Nie tylko abstrakcyjne metody.
  - Klasy mogą dziedziczyć tylko po jednej klasie abstrakcyjnej.
- Interfejs
  - Służy do opisu zachowania.
  - Tylko abstrakcyjne metody.
  - Klasy mogą implementować wiele interfejsów.

Metody domyślne

- Od Javy 8 wprowadzono do interfejsów metody domyślne.
- Są to domyślne implementacje metod z interfejsu.

Metoda domyślna interfejsu `Comparable`

```
1  public interface Comparable<T>{
2    default int compareTo(T other) { return 0; }
3  // domyslnie wszystkie elementy sa takie same
4  }
```

- Metody domyślne pozwalają na nadpisanie tylko wybranych metod w klasie implementującej, pozostałe będą funkcjonowały w sposób domyślny.
- Metody domyślne powodują pewne komplikacje w funkcjonowaniu hierarchii interfejsów.

Przesłanianie metod domyślnych

1. Implementacja metody w klasie zawsze przesłania implementację domyślną.
2. Jeżeli klasa implementuje interfejsy z metodą o tej samej nazwie i argumentach i przynajmniej jedna z nich ma implementację domyślną to klasa musi nadpisać tę metodę.

Printable

```
public interface Printable {
    Color[] print();
    default int count() {return 1;}
}
```

Countable

```
public interface Countable {
    int count();
}
```

Banknote

```
public class Banknote implements Printable, Countable {
  static Color green = Color.green;
  static Color white = Color.white;
  @Override
  public Color[] print() {
      return new Color[] =
          {green,green,green,green,white,green,green,green,green} ;
    }
  @Override
  public int count() { return 2;}
}
```

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.