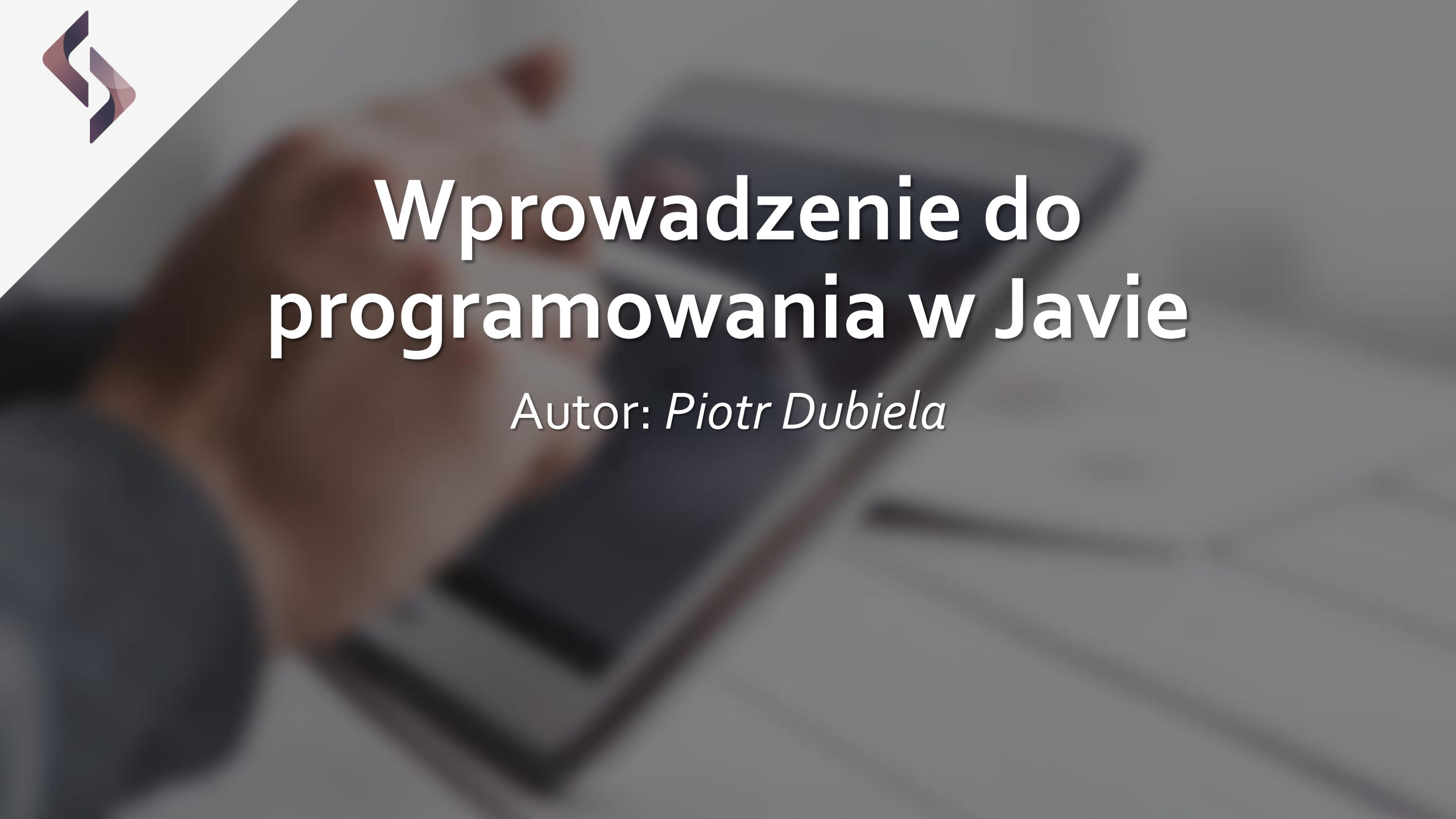
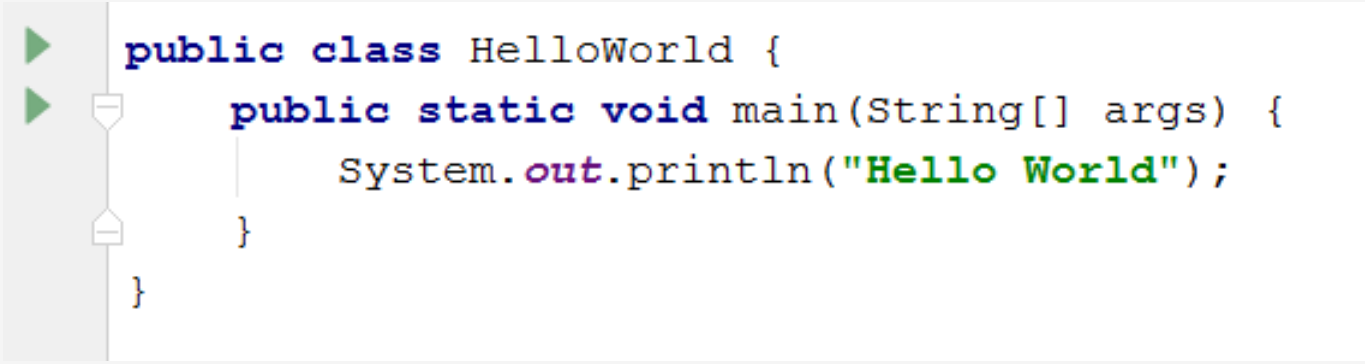


Wprowadzenie do programowania w Javie

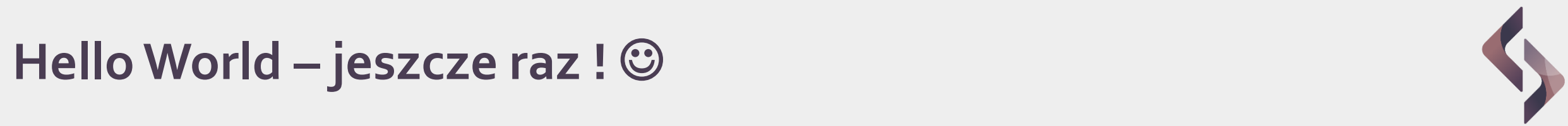
Autor: *Piotr Dubiela*

Hello World – jeszcze raz ! ☺



```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

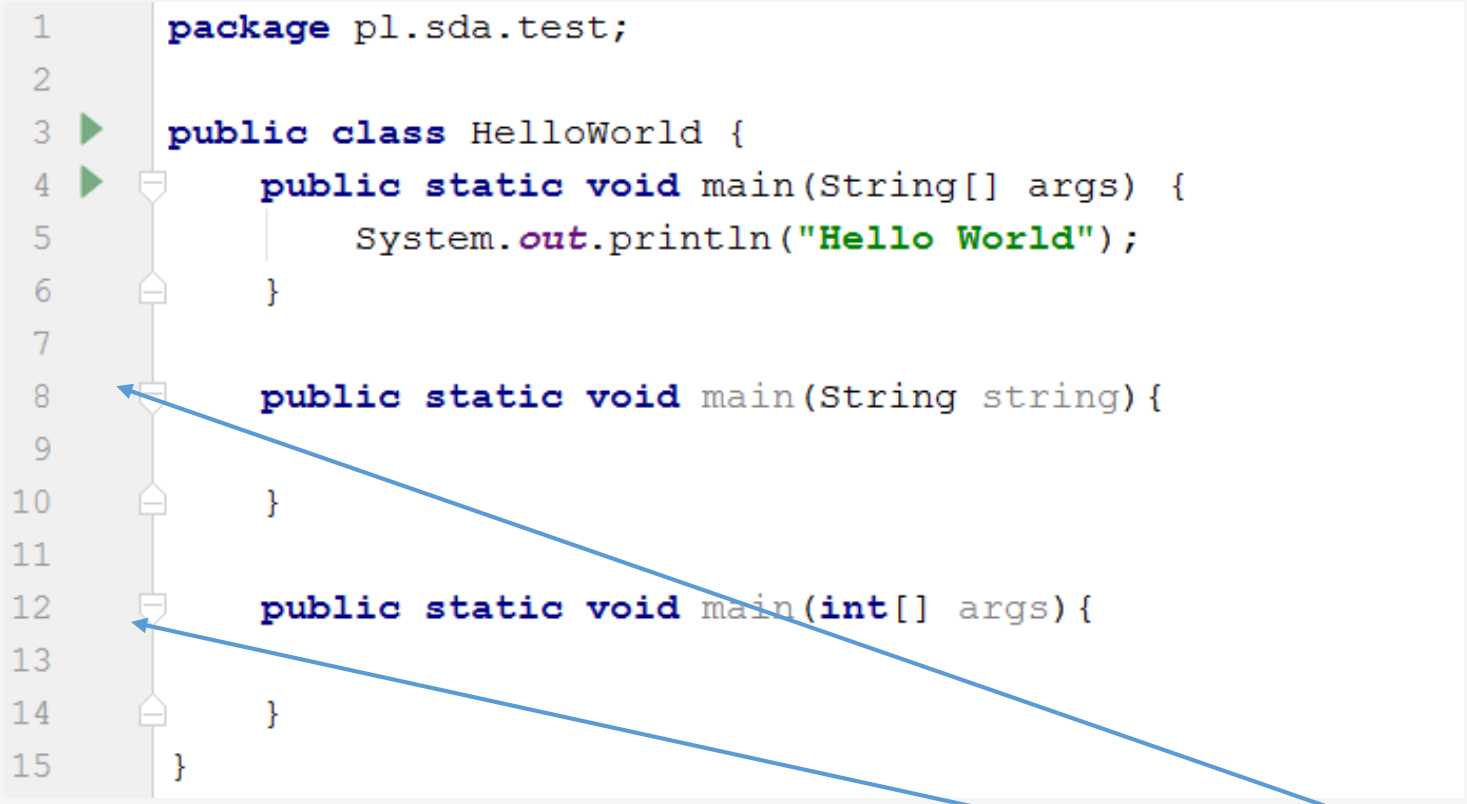
Hello World – jeszcze raz ! ☺



play oznacza, że możemy uruchomić program przez uruchomienie pojedynczej metody

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Hello World – jeszcze raz ! ☺



```
1   package pl.sda.test;
2
3   public class HelloWorld {
4       public static void main(String[] args) {
5           System.out.println("Hello World");
6       }
7
8       public static void main(String string){
9
10      }
11
12      public static void main(int[] args){
13
14      }
15  }
```

Pomimo tych samych modyfikatorów i nazw argumentów oraz nazwy metody żadna z metod nie otrzymała przycisku ‚play'

Hello World – jeszcze raz ! ☺

Metoda ‚main' - publiczna, statyczna, bez typu zwracanego, przyjmująca tablicę znaków stanowi punkt wejścia programu w Javie

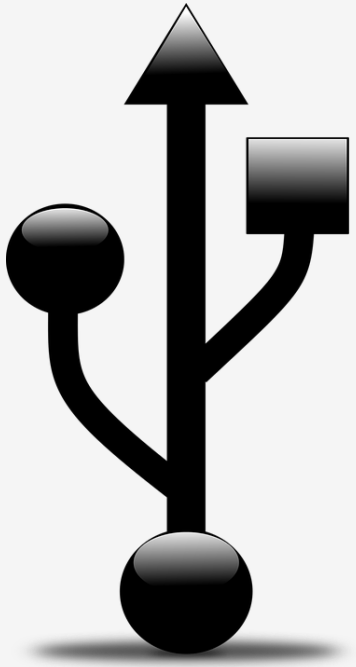
```
1   package pl.sda.test;
2
3   public class HelloWorld {
4       public static void main(String[] args) {
5           System.out.println("Hello World");
6       }
7
8       public static void main(String string){
9
10      }
11
12      public static void main(int[] args){
13
14      }
15  }
```

Hello World – jeszcze raz ! ☺

Czym jest zatem argument punktu wejścia ?

- Zbiór tzw. danych wejściowych dla programu
- Pozwala na automatyczne wykorzystanie programu np. program do rozsyłania powiadomień mailowych, uruchamiający się cyklicznie dla zadanych parametrów np. o 10 powiadomienia dla grupy A, o 11 dla grupy B itp.
- Maksymalnie jeden punkt wejścia w 1 klasie

Interfejsy



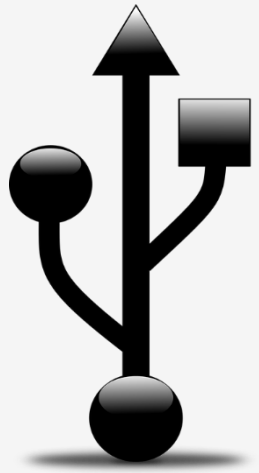
Czym jest interfejs w Javie?

Interfejs (*Interface*) – abstrakcyjny typ, posiadający jedynie operacje i nie posiadający danych.

Interfejsy

Cechy Interfejsu:

- Podobny do klasy abstrakcyjnej, ale nie można zadeklarować konstruktora
- Wszystkie metody są domyślnie publiczne i abstrakcyjne
- Interfejs może rozszerzać jeden lub wiele innych interfejsów
- Klasa może *implementować* wiele interfejsów
- Obiekty klasy implementującej interfejs możemy rzutować na typ Interefejsu
- Interfejs może posiadać jedynie finalne statyczne pola

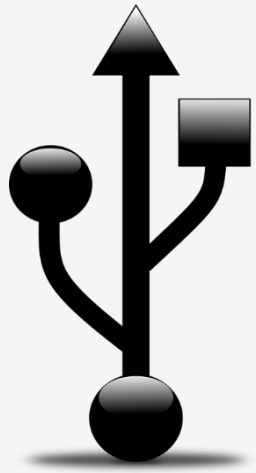


Interfejsy

Interfejs:

- Przykładowy interfejs:

```
3  public interface Ruchowy {
4      public void doGory();
5      public abstract void wDol();
6      void wLewo();
7      void wPrawo();
8  }
```



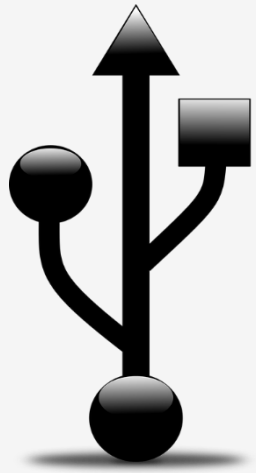
Interfejsy

Interfejs:

- Przykładowy interfejs:

```
3  public interface Ruchowy {
4      public void doGory();
5      public abstract void wDol();
6      void wLewo();
7      void wPrawo();
8  }
```

Modyfikatory są bez znaczenia
Każda metoda jest domyślnie public abstract

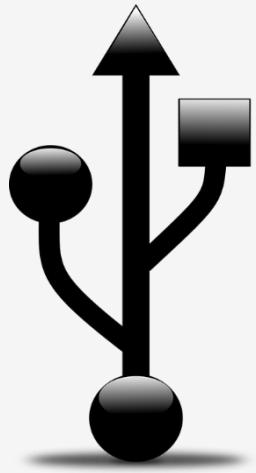


Interfejsy

Interfejs:

- Przykładowa implementacja interfejsu:

```
10  class Gracz implements Ruchowy {
11      int x;
12      int y;
13      int predkosc;
14
15      public Gracz(int predkosc){
16          this.predkosc = predkosc;
17      }
18
19      @Override
20      public void doGory() {
21          x+=predkosc;
22      }
23
24      @Override
25      public void wDol() {
26          y-=predkosc;
27      }
28
29      @Override
30      public void wLewo() {
31          x-=predkosc;
32      }
33
34      @Override
35      public void wPrawo() {
36          x+=predkosc;
37      }
38  }
```



Interfejsy – zadanie

1. *Utwórz interfejs Instrumentalny*
2. *Dodaj metodę graj():void, która wyświetli dźwięk grania w formie tekstu*
3. *Utwórz klasy Bęben, Gitara, Pianino*
4. *Zaimplementuj interfejs w klasach*
5. *Przetestuj działanie tworząc po 1 obiekcie z każdej klasy*

Interfejsy – zadanie 2

1. *Utwórz interfejs Dzwoni*
2. *Dodaj mu pole statyczne dla przechowywania numeru alarmowego*
3. *Dodaj metody:*
    1. *zadzwon(String):void*
    2. *zadzwonNaNrAlarmowy():void*
4. *Utwórz klasę Telefon o polach:*
    1. *numerTelefonu: String*
    2. *lacznyCzasRozmow: int*
5. *Zaimplementuj interfejs Dzwoni w Telefonie*
6. *Niech zadzwon() losowo się nie dodzwania na wybrany numer*
7. *Przetestuj rozwiązanie*
8. ** zadzwon() niech generuje losowy czas rozmowy w zakresie 1 minuty do godziny*
9. ** Podsumowanie czasu rozmowy powinno się wyświetlić pod koniec metody zadzwon*

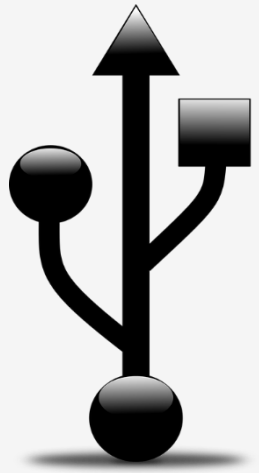
Interfejsy – zadanie 3

1. *Utwórz klasę Pokarm o polach:*
    1. *Nazwa*
    2. *TypPokarmu*
    3. *Waga*
2. *Utwórz enum TypPokarmu {MIĘSO, OWOCE, NABIAŁ...}*
3. *Utwórz interfejs Jedzący*
4. *Dodaj metody*
    1. *jedz(Pokarm pokarm):void*
    2. *ilePosilkowZjedzone(): int*
    3. *ileGramowZjedzone() :int*
5. *Utwórz klasy Weganin, Krokodyl, Programista*
6. *Zaimplementuj interfejs w wymienionych klasach*
7. *Uwzględnij typ jedzenia oraz możliwości osobników*
8. *W klasie Main utwórz przedstawicieli klas i dodaj ich do wspólnej tablicy*
9. *Przeiteruj tablicę kilkukrotnie dla różnych pokarmów i znajdź 2 zwycięzców*
    1. *Gracza który zjadł najwięcej posiłków (Pokarmów)*
    2. *Gracza ktory zjadł największą masę jedzenia (gramy)*

Interfejsy

Nowość w Javie 8:

- Dodano możliwość deklarowania domyślnych metod (*default*), dzięki czemu możemy dodać metodę dla wielu klas implementujących interfejs z domyślnym blokiem kodu
- Dodano metody statyczne, posiadające kod, nie wymagające nadpisywania

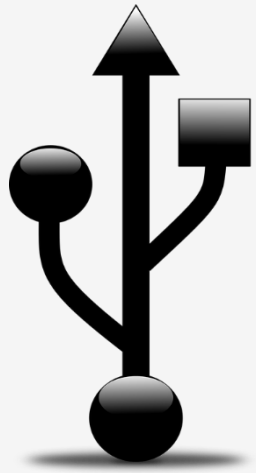


Interfejsy

Metoda default:

- Załóżmy, że posiadamy poniższy interfejs:

```
3  public interface Powiekszalny {
4      int pobierzSzerokosc();
5      int pobierzWysokosc();
6
7      void powieksz0(int wymiar);
8      void poszerz0(int wymiar);
9  }
```



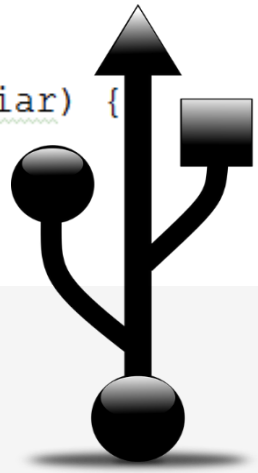
Interfejsy

Metoda default:

- Oraz wiele implementacji tego typu:

```
11  class Okreg implements Powiekszalny{
12      private int szerokosc;
13      private int wysokosc;
14
15      public Okreg(int szerokosc, int wysokosc) {
16          this.szerokosc = szerokosc;
17          this.wysokosc = wysokosc;
18      }
19
20      @Override
21      public int pobierzSzerokosc() {
22          return this.szerokosc;
23      }
```

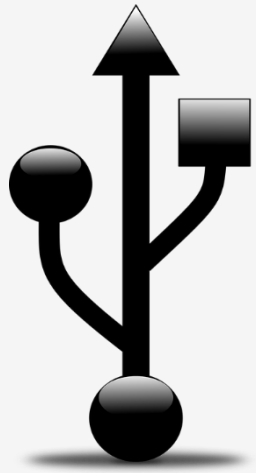
```
25      @Override
26      public int pobierzWysokosc() {
27          return this.wysokosc;
28      }
29
30      @Override
31      public void powieksz0(int wymiar) {
32          this.wysokosc+=wymiar;
33      }
34
35      @Override
36      public void poszerz0(int wymiar) {
37          this.szerokosc+=wymiar;
38      }
39  }
```



Interfejsy

Metoda default:

- Przed JDK 1.8 w celu dodania metody powiekszNRazy należałoby zaktualizować wszystkie klasy implementujące dany interfejs

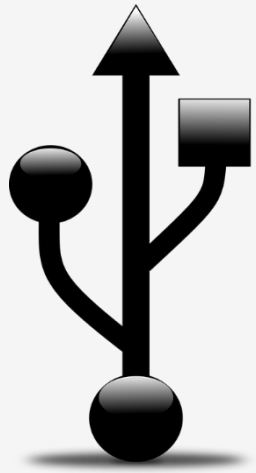


Interfejsy

Metoda default:

- Od JDK 1.8 wystarczy dodanie metody default:

```
3   public interface Powiekszalny {
4       int pobierzSzerokosc();
5       int pobierzWysokosc();
6
7       void powieksz0(int wymiar);
8       void poszerz0(int wymiar);
9
10      default void powiekszNRazy(int n){
11          int aktualnaWysokosc = pobierzWysokosc();
12          for (int i=1; i<n; i++){
13              powieksz0(aktualnaWysokosc);
14          }
15      }
16
17      default void poszerzNRazy(int n){
18          int aktualnaSzerokosc = pobierzSzerokosc();
19          for (int i=1; i<n; i++){
20              poszerz0(aktualnaSzerokosc);
21          }
22      }
23  }
```



Interfejsy – zadanie 4

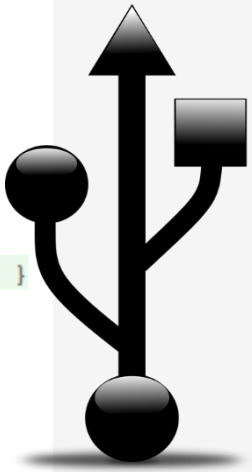
1. *Utwórz interfejs Chłodzi:*
    1. *pobierzTemp():double*
    2. *schlodz():void*
2. *Utwórz interfejs Grzeje:*
    1. *pobierzTemp():double*
    2. *zwiekszTemp():void*
3. *Utwórz 3 klasy: Farelka(Grzeje), Wiatrak(Chłodzi), Klimatyzacja(Grzeje, Chłodzi)*
4. *Przetestuj działanie w klasie Main*
5. *Dodaj metodę default wyswietlTemp():void w obu interfejsach, któ wypisuje tekst: „Aktualna temperatura w pomieszczeniu wynosi xx.x stopni Celsjusza”*
6. ** Rozwiąż konflikt dla Klimatyzacji*

Interfejsy

Comparable:

- Interfejs pozwalający na sortowanie obiektów dowolnego typu
- Metoda compareTo() powinna zwrócić 0 jeśli obiekty są sobie równe

```
5  public class Czlowiek implements Comparable{
6      String imie;
7      String nazwisko;
8      Plec plec;
9
10     public Czlowiek(String imie, String nazwisko, Plec plec) {
11         this.imie = imie;
12         this.nazwisko = nazwisko;
13         this.plec = plec;
14     }
15
16     public Plec pobierzPlec() { return plec; }
19
20     @Override
21     public String toString() { return String.format("%s : %s %s", plec, imie, nazwisko); }
24
25     @Override
26     public int compareTo(Object o) {
27         Czlowiek that = (Czlowiek)o;
28         return nazwisko.compareTo(that.nazwisko);
29     }
```

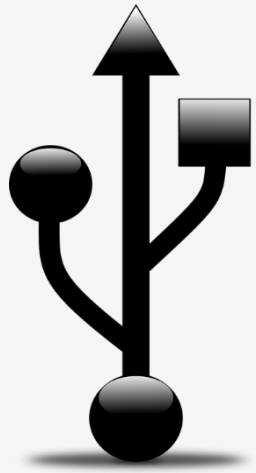


Interfejsy

Comparable:

- Użycie:

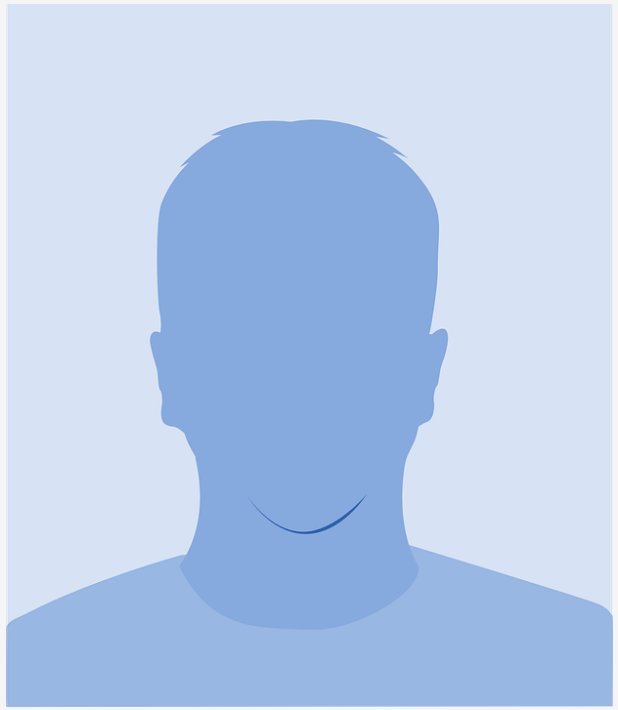
```
31      public static void main(String[] args) {
32          Czlowiek adam = new Czlowiek( imie: "Adam",  nazwisko: "Kowalski", Plec.MEZCZYZNA);
33          Czlowiek malgorzata = new Czlowiek( imie: "Agnieszka",  nazwisko: "Adamowicz", Plec.KOBIETA);
34          Czlowiek panSamolot = new Czlowiek( imie: "Jerzy",  nazwisko: "Rokicki", Plec.SAMOLOT);
35
36          Czlowiek[] ludki = new Czlowiek[]{adam, malgorzata, panSamolot};
37          wyswietlLudkow(ludki);
38          System.out.println("Sortowanie ");
39          Arrays.sort(ludki);
40          wyswietlLudkow(ludki);
41      }
42
43      private static void wyswietlLudkow(Czlowiek[] ludki) {
44          for(Czlowiek ludek:ludki){
45              System.out.println(ludek);
46          }
47      }
```



Interfejsy – zadanie 5

1. *Utwórz klasę Student*
2. *Nadaj następujące atrybuty: Imie, Nazwisko, numer albumu*
3. *Zaimplementuj interfejs Comparable, tak aby sortować studentów od najmniejszego numeru indeksu do największego*
4. *W metodzie psvm utwórz kilka obiektów typu Student i dodaj do tablicy*
5. *Wyświetl Studentów przed i po sortowaniu*
6. ** Zrób odwrotne sortowanie*

Typy generyczne



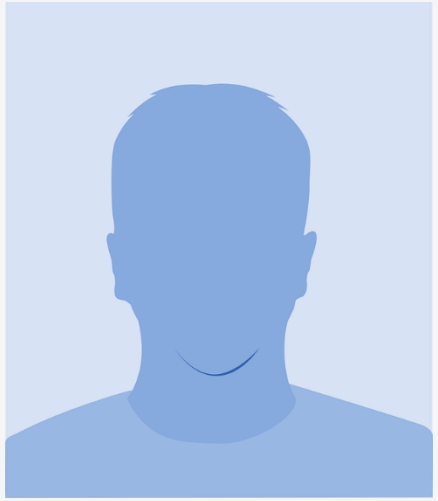
Czym są typy generyczne ?

Programowanie uogólnione (generyczne) – paradygmat programowania, pozwalający na pisanie kodu programu bez wcześniejszej znajomości typów danych, na których ten kod będzie pracował.

Typy generyczne

Po co stosować?

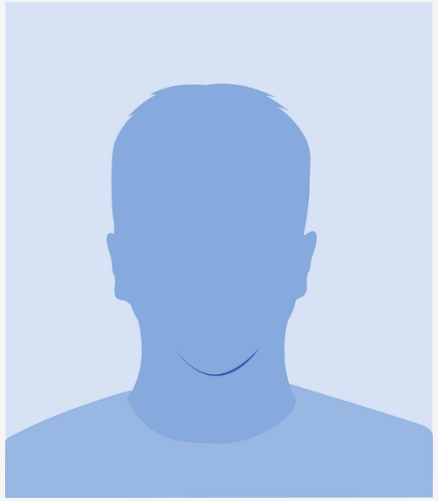
- Zapewniają lepszą kontrolę typów oraz unikanie rzutowania
- Pozwalają na wygodniejszą implementację algorytmów (jedna implementacja dla różnych typów danych)
- Podajemy typ danych dopiero w momencie użycia



Typy generyczne

Przykłady zastosowania w Javie:

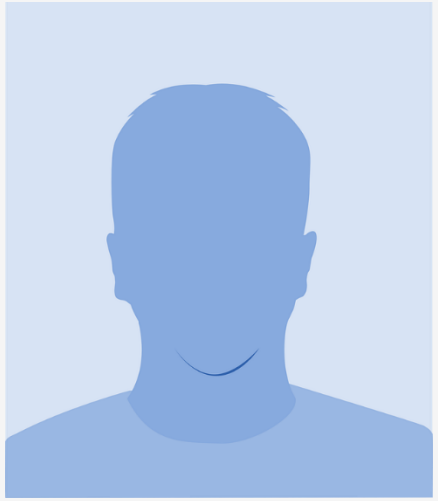
- Collection <E>
- Comparable <T>
- Map <K,V>
- Optional <T>



Typy generyczne

Załóżmy że chcemy mieć obiekt przechowujący parę innych obiektów:

```
6  public class Para {
7      private Object lewy;
8      private Object prawy;
9
10     public Para(Object lewy, Object prawy) {
11         this.lewy = lewy;
12         this.prawy = prawy;
13     }
14
15     public Object wezLewy(){
16         return lewy;
17     }
18
19     public Object wezPrawy(){
20         return prawy;
21     }
22
23     @Override
24     public String toString() {
25         return String.format("[%s ; %s]", lewy, prawy);
26     }
```

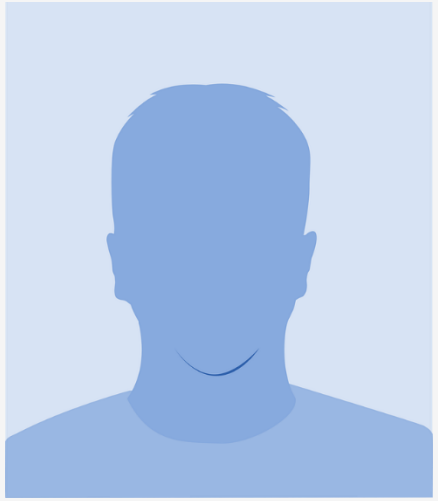


Typy generyczne

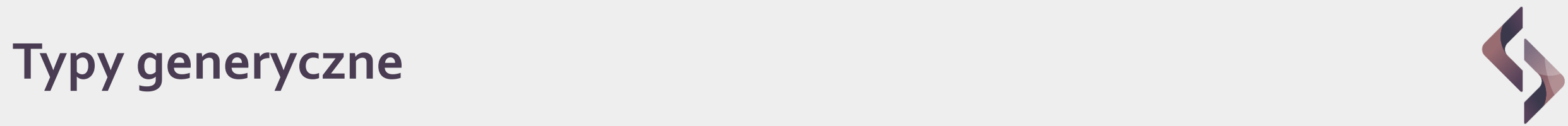
Tworzymy 2 obiekty typu człowiek i umieszczamy w obiekcie Pary:

```
28      public static void main(String[] args) {
29          Czlowiek monika = new Czlowiek( imie: "Monika",  nazwisko: "Bogdan", Plec.KOBIETA);
30          Czlowiek tomasz = new Czlowiek( imie: "Tomasz",  nazwisko: "Nowak", Plec.MEZCZYZNA);
31          Para para = new Para(monika, tomasz);
32          System.out.println(para);
33
34      }
```

[Kobieta : Monika Bogdan ; Mezczyzna : Tomasz Nowak]



Typy generyczne

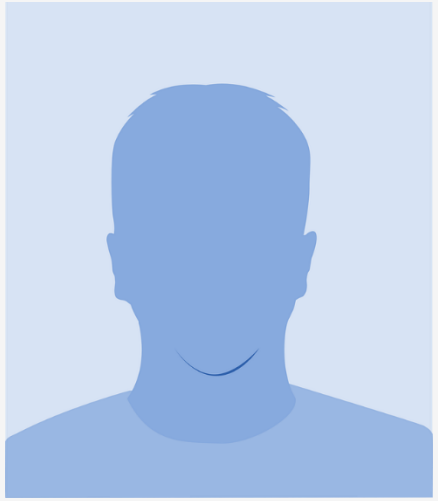


Problem pojawia się gdy chcemy użyć metody dostępnej na klasie Człowiek:

```
    public static void main(String[] args) {
        Czlowiek monika = new Czlowiek( imie: "Monika",  nazwisko: "Bogdan", Plec.KOBIETA);
        Czlowiek tomasz = new Czlowiek( imie: "Tomasz",  nazwisko: "Nowak", Plec.MEZCZYZNA);
        Para para = new Para(monika, tomasz);
        System.out.println(para);
        para.wezPrawy().
    }
}
```

```
m equals(Object obj)                          boolean
m hashCode()                                      int
m toString()                                   String
m getClass()                      Class<? extends Object>
m notify()                                        void
m notifyAll()                                     void
m wait()                                          void
m wait(long timeout)                              void
m wait(long timeout, int nanos)                   void
  cast                                ((SomeType) expr)
Ctrl+Down and Ctrl+Up will move caret down and up in the editor >>
```

Otrzymujemy jedynie dostęp do metod klasy Object



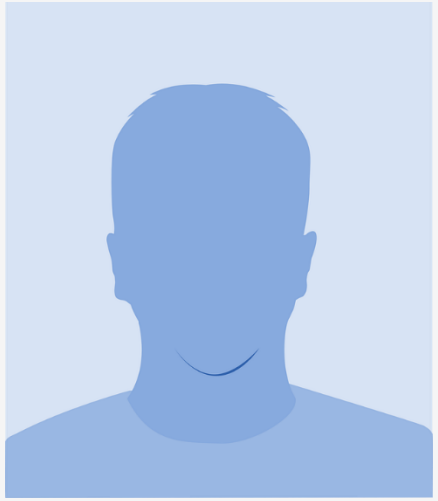
Typy generyczne

Możemy obejść to poprzez rzutowanie:

Każdorazowe rzutowanie byłoby nieprzyjemne w użyciu i zmniejszało czytelność kodu

```
28      public static void main(String[] args) {
29          Czlowiek monika = new Czlowiek( imie: "Monika",  nazwisko: "Bogdan", Plec.KOBIETA);
30          Czlowiek tomasz = new Czlowiek( imie: "Tomasz",  nazwisko: "Nowak", Plec.MEZCZYZNA);
31          Para para = new Para(monika, tomasz);
32          System.out.println(para);
33          Czlowiek lewy = (Czlowiek)para.wezLewy();
34          lewy.
35      }
36  }
37
```

```
compareTo(Object o)                    int
pobierzPlec()                          Plec
toString()                             String
```

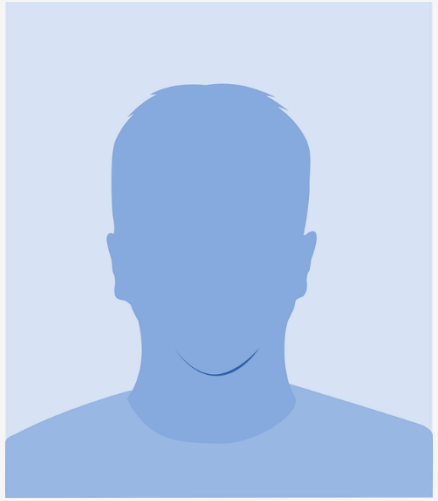


Typy generyczne

Rozwiązanie - generyki:

```
6  public class Para <T>{
7      private T lewy;
8      private T prawy;
9
10     public Para(T lewy, T prawy) {
11         this.lewy = lewy;
12         this.prawy = prawy;
13     }
14
15     public T wezLewy(){
16         return lewy;
17     }
18
19     public T wezPrawy(){
20         return prawy;
21     }
22
23     @Override
24     public String toString() {
25         return String.format("[%s ; %s]", lewy, prawy);
26     }
```

Deklaracja dowolnego typu T

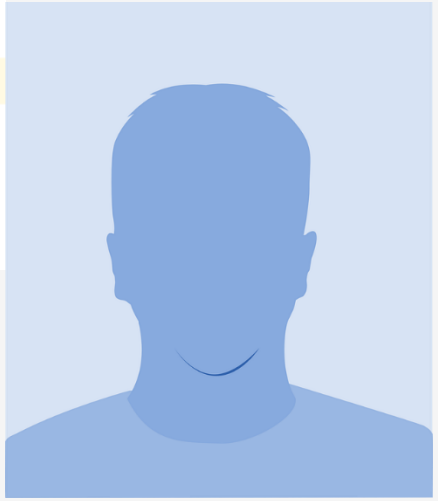


Typy generyczne

Rozwiązanie - generyki:

```
28      public static void main(String[] args) {
29          Czlowiek monika = new Czlowiek( imie: "Monika",  nazwisko: "Bogdan", Plec.KOBIETA);
30          Czlowiek tomasz = new Czlowiek( imie: "Tomasz",  nazwisko: "Nowak", Plec.MEZCZYZNA);
31          Para<Czlowiek> para = new Para<Czlowiek>(monika, tomasz);
32          System.out.println(para);
33          para.wezPrawy().
34      }                    pobierzPlec()                                  Plec
35  }                        compareTo(Object o)                            int
36                           toString()                                     String
```

Deklaracja przechowywania obiektów typu Czlowiek



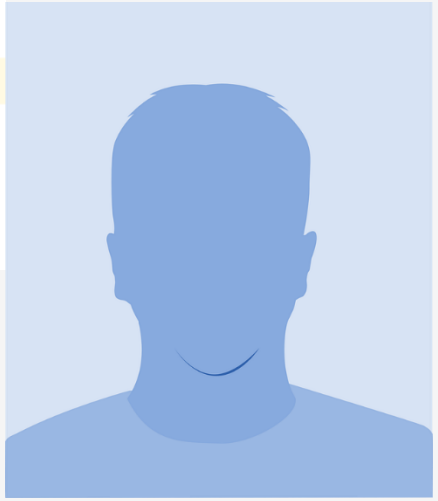
Typy generyczne

Rozwiązanie - generyki:

```
28      public static void main(String[] args) {
29          Czlowiek monika = new Czlowiek( imie: "Monika",  nazwisko: "Bogdan", Plec.KOBIETA);
30          Czlowiek tomasz = new Czlowiek( imie: "Tomasz",  nazwisko: "Nowak", Plec.MEZCZYZNA);
31          Para<Czlowiek> para = new Para<Czlowiek>(monika, tomasz);
32          System.out.println(para);
33          para.wezPrawy().
34      }
35  }
36
```

```
pobierzPlec()                    Plec
compareTo(Object o)              int
toString()                       String
```

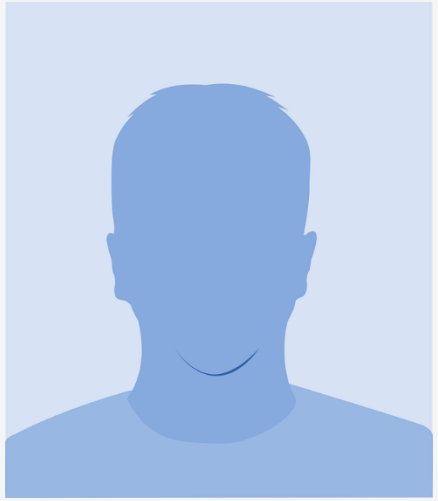
Otrzymujemy dostęp do metod klasy człowiek !



Typy generyczne

Konwencja nazewnicza typu danych:

- **E** – element
- **K** – klucz
- **N** – liczba
- **T** – typ
- **V** – wartość (value)
- **S, U, V** – kolejne n-te Typy

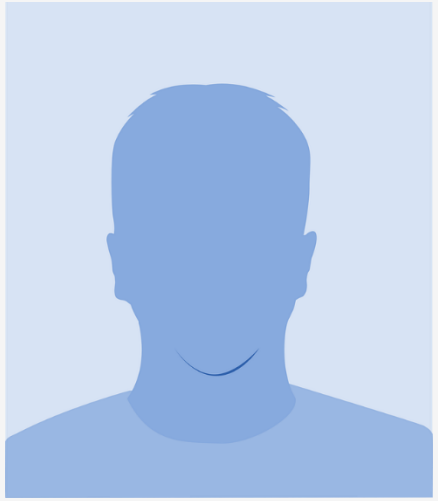


Typy generyczne

Ograniczanie typów – odgórne *(upper bound):*

- Akceptujemy jedynie klasy które będą dziedziczyć po innej klasie
- W klasie :

```
6  public class Para <T extends Czlowiek>{
7      private T lewy;
8      private T prawy;
9
```

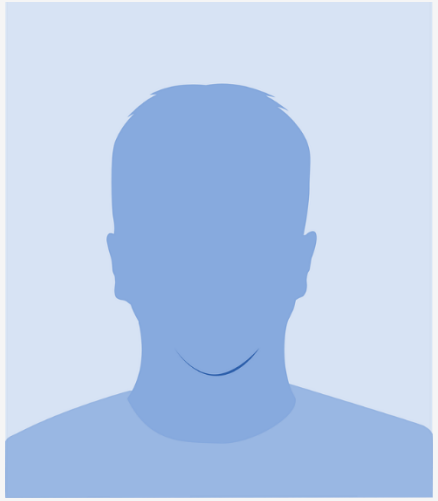


Typy generyczne

Ograniczanie typów – odgórne *(upper bound):*

- Akceptujemy jedynie klasy które będą dziedziczyć po innej klasie
- W metodzie :

```
13      public static void zamienParke(Para<? extends Czlowiek> ludki){
14          System.out.println(ludki);
15          ludki.zamien();
16          System.out.println(ludki);
17      }
```

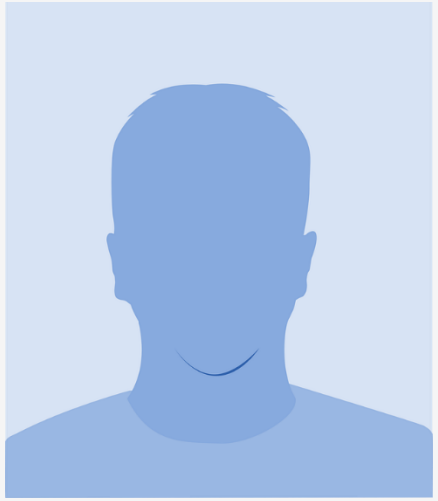


Typy generyczne

Ograniczanie typów – oddolne *(lower bound):*

- Akceptujemy jedynie klasy które znajdują się nad wybraną klasą (superklasy wybranej klasy)
- Zakładając, że Student i Weganin dziedziczą bezpośrednio po klasie Człowiek:

```
14          Para<Student> ludki = new Para(zuzia, stasiu);
15          Para<Weganin> ludki2 = new Para(alicja, stasiu);
16
17          zamienParke(ludki);
18  //        zamienParke(ludki2); błąd kompilacji
19      }
20
21      public static void zamienParke(Para<? super Student> ludki){
22          System.out.println(ludki);
23          ludki.zamien();
24          System.out.println(ludki);
25      }
```

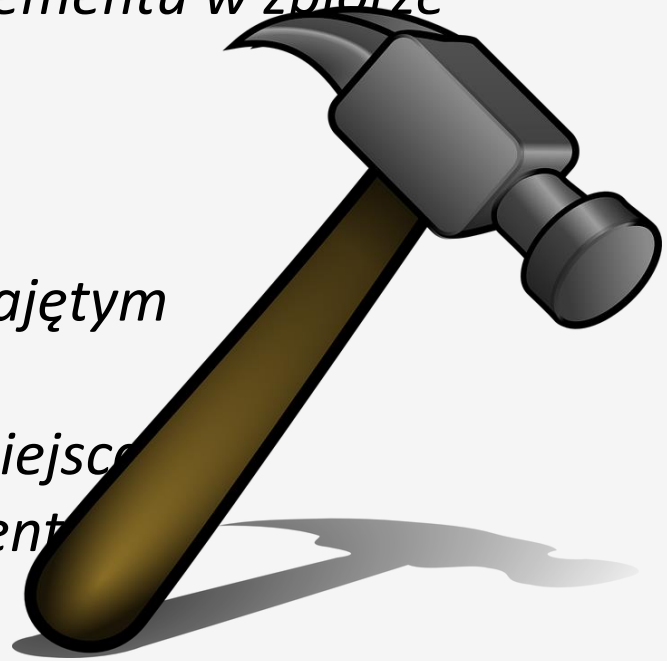


Typy generyczne – zadanie 1

1. *Utwórz klasę generyczną Garaż, która umożliwia przechowywanie 2 Samochodów*
2. *Utwórz klasę Samochod z polami:*
    1. *Marka*
    2. *Model*
    3. *Kolor*
3. *Oraz metodami:*
    1. *toString() – zwracające opis auta*
    2. *zmieńKolor(String kolor)*
4. *Utwórz klasy BMW i Porshe dziedziczące po Samochodzie z mniejszą liczbą argumentów*
5. *Dodaj metody do klasy Garaż:*
    1. *zaparkuj(auto) :void*
    2. *wyprowadz(auto):auto*
6. ** Utwórz własny wyjątek na sytuację gdy oba miejsca są już zajęte i nie można zaparkować kolejnego auta*

Typy generyczne – zadanie 2

1. *Utwórz klasę generyczną MojaLista pozwalającą na przechowywanie n elementów*
2. *Użyj tablicy do przechowywania elementów*
3. *Dodaj konstruktor MojaLista(int n) – który określi max ilość elementów*
4. *Dodaj metodę zawiera(E element):boolean*
    1. *Zwróci true ⇔ element jest equals względem innego elementu w zbiorze*
5. *Dodaj metodę rozmiar():int*
6. ** Niech rozmiar() zwróci zajętą ilość elementów*
7. *Dodaj metodę dodaj(E element):boolean*
    1. *Element zostaje dodany w wolne miejsce po ostatnim zajętym elemencie*
    2. *Zwraca true jeśli udało się dodać i false jeśli zabrakło miejsca*
8. *Dodaj metodę toString():String zwracającą wszystkie elementy w formie opisowej, odseparowane przecinkami: [element1, element2, element3 ...]*



Kolekcje

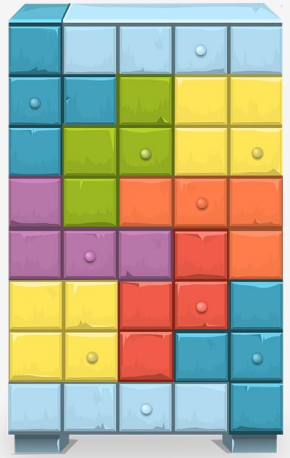
Czym są kolekcje ?

Kolekcje *(collections)*– są to specjalne klasy w Javie przeznaczone do przechowywania zbiorów obiektów, tworzących w ten sposób zestaw danych, na których możemy wykonywać operacje oraz przeglądać poszczególne elementy

Kolekcje

Cechy kolekcji:

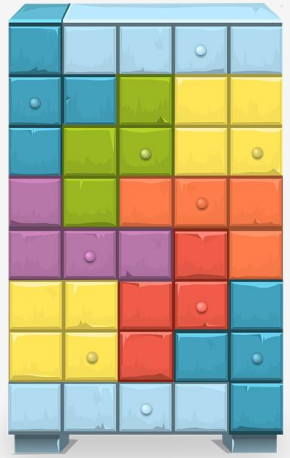
- W skrócie mówimy o nich jako ‚tablicach na sterydach'
- Stanowią struktury danych, które mogą działać lepiej lub gorzej w zależności od ich użycia
- Mogą zachowywać kolejność elementów lub nie
- Rozmiar kolekcji jest dynamiczny (w przeciwieństwie do tablic)
- Stanowią podzbiór pakietu *java.util.Collection*



Kolekcje

Składniki kolekcji:

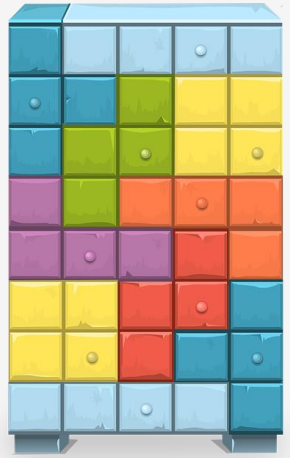
- Interfejsy
  - Abstrakcyjne typy danych reprezentujące kolekcje
  - Opisują sposób korzystania z kolekcji niezależny od implementacji
  - Przykłady List, Set, Map



Kolekcje

Składniki kolekcji:

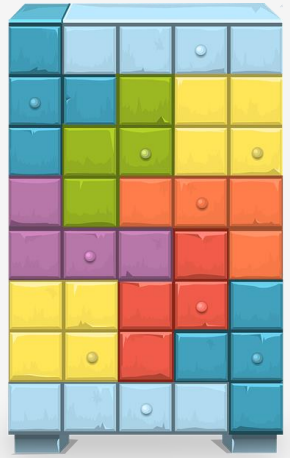
- Interfejsy
- Implementacje
  - Konkretne implementacje interfejsów (klasy użytkowe)
  - np. ArrayList, HashMap, HashSet



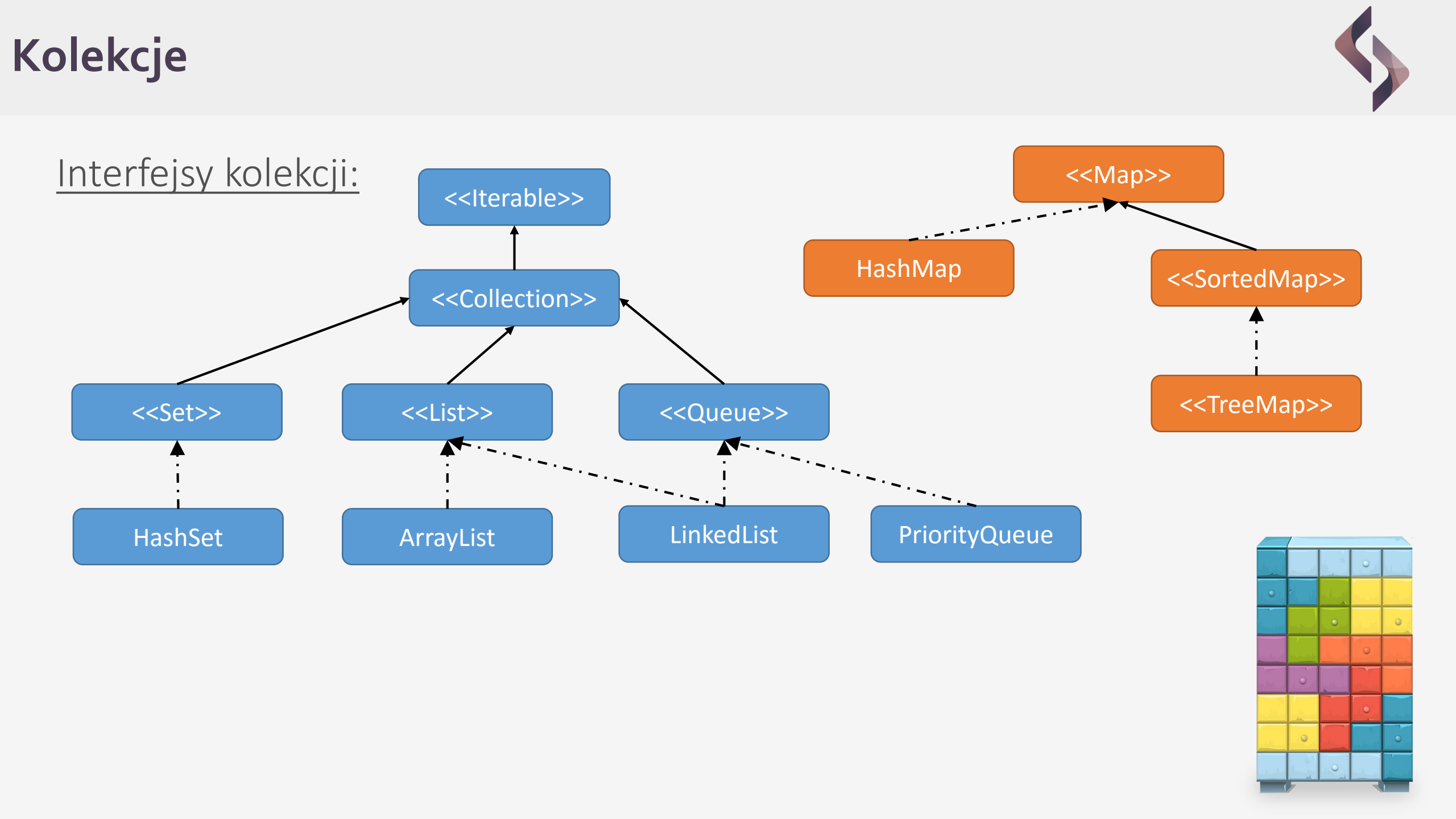
Kolekcje

Składniki kolekcji:

- Interfejsy
- Implementacje
- Algorytmy
  - Metody realizacji operacji na kolekcjach takich jak wyszukiwanie, sortowanie
  - Są polimorficzne i działają dla różnych implementacji danego interfejsu kolekcji



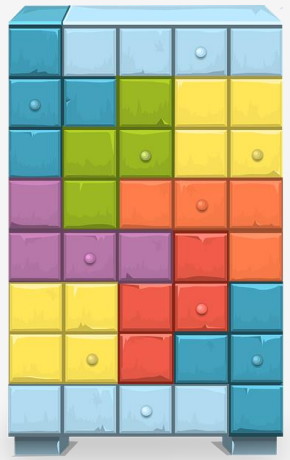
Kolekcje


Interfejsy kolekcji:
<<Iterable>>
<<Collection>>
<<Set>>
<<List>>
<<Queue>>
HashSet
ArrayList
LinkedList
PriorityQueue
<<Map>>
HashMap
<<SortedMap>>
<<TreeMap>>

Kolekcje

Lista:

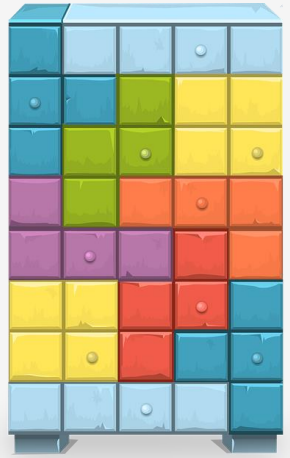
- Odpowiednik tablicy z dynamicznym rozmiarem
- Zapewnia kolejność elementów
- Ten sam obiekt może być elementem kolekcji wielokrotnie



Kolekcje

Lista – metody interfejsu List<E>:

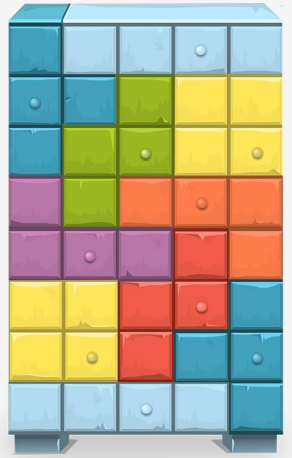
- *add(E e) – dodaje element na koniec listy*
- *add(E e, int index) – dodaje element na wybraną pozycję (nie usuwa elementu już tam się znajdującego)*
- *addAll(Collection) – dodaje kolekcję na końcu listy*
- *contains(E e) – zwraca true jeśli element znajduje się w liście*
- *get(int index) – zwraca element o wybranym indeksie*
- *indexOf(E e) – zwraca indeks elementu w liście (1 wystąpienie)*
- *isEmpty() – zwraca true jeśli lista jest pusta*
- *lastIndexOf(E e) – zwraca indeks ostatniego wystąpienia elementu w liście*
- *remove(E e) – usuwa pierwsze wystąpienie wskazanego elementu*
- *remove (int index) – usuwa element pod wskazanym indeksem*
- *set(E e, int index) – wstawia element na wybraną pozycję (tym samym poprzednio znajdujący się element zostaje usunięty)*
- *size() – zwraca rozmiar listy*



Kolekcje

Lista – implementacje interfejsu:

- ArrayList
  - Używana w większości przypadków
  - Preferowana gdy częściej szukamy i odwołujemy się do obiektów (O(1))
- LinkedList
  - Pozwala na szybszą manipulację danymi (dodawanie, usuwanie elementów)
  - Może być traktowana jako List lub Queue, ponieważ implementuje oba te interfejsy



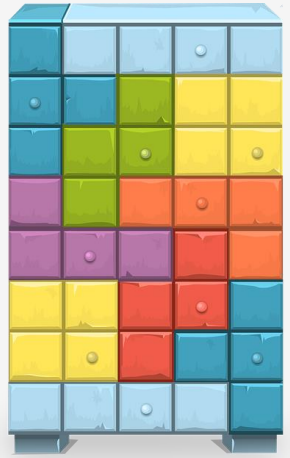
Kolekcje

Tworzenie kolekcji typu Lista – dobra praktyka:

- Odwołując się do kolekcji powinniśmy preferować użycie interfejsu zamiast konkretnego typu
- Pozwala to na większą generalizację i lepszą użytkowość tworzonych metod

```
LinkedList<String> lista = new LinkedList<>();
```

```
private static void wyswietlListe(LinkedList lista){
    System.out.println(lista);
}
```



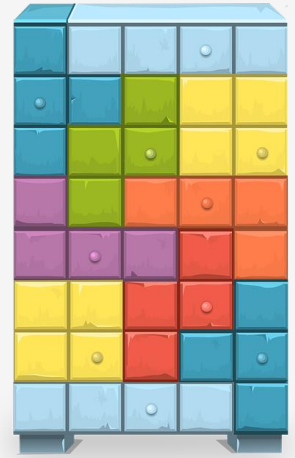
Kolekcje

Tworzenie kolekcji typu Lista – dobra praktyka:

- Odwołując się do kolekcji powinniśmy preferować użycie interfejsu zamiast konkretnego typu
- Pozwala to na większą generalizację i lepszą użytkowość tworzonych metod

```
LinkedList<Str          w LinkedList<>();
```

```
private stat  oid wysw      (LinkedList lista){
    System.out
}
```



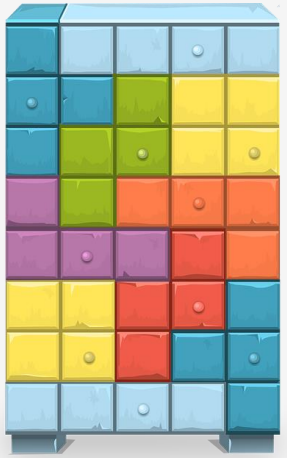
Kolekcje

Tworzenie kolekcji typu Lista – dobra praktyka:

- Odwołując się do kolekcji powinniśmy preferować użycie interfejsu zamiast konkretnego typu
- Pozwala to na większą generalizację i lepszą użytkowość tworzonych metod

```
List<String> lista = new LinkedList<>();
```

```
private static void wyswietlListe(List lista){
    System.out.println(lista);
}
```



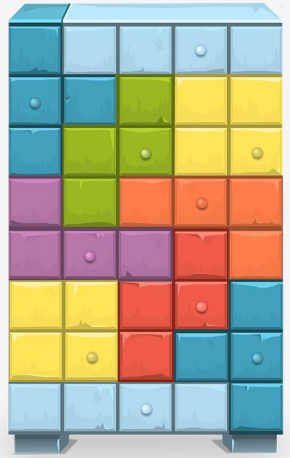
Kolekcje

Tworzenie kolekcji typu Lista – dobra praktyka:

- Odwołując się do kolekcji powinniśmy preferować użycie interfejsu zamiast konkretnego typu
- Pozwala to na większą generalizację i lepszą użytkowość tworzonych metod

```
List<String> lista = new LinkedList<>();
```

```
private static void wyswietlListe(List lista){
    System.out.println(lista);
}
```



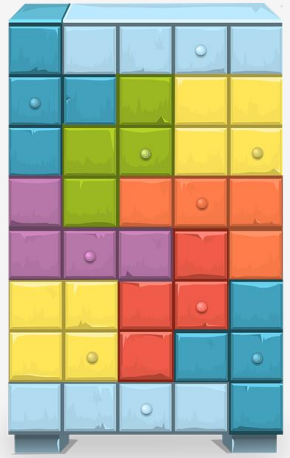
Kolekcje – zadanie 1

1. *Utwórz listę kilku elementów typu String, a następnie prześledź:*
    1. *Działanie metody add(E e)*
    2. *Działanie metody set(E e, int index)*
    3. *Działanie metody indexOf(Object o)*
    4. *Działanie metody lastIndexOf(Object o)*
    5. *Działanie metody remove(Object o)*
    6. *Działanie metody remove(index int)*
2. *Utwórz metodę do wyświetlania zduplikowanych elementów w liście*
3. *Utwórz metodę do usuwania zduplikowanych elementów w liście*

Kolekcje

Set:

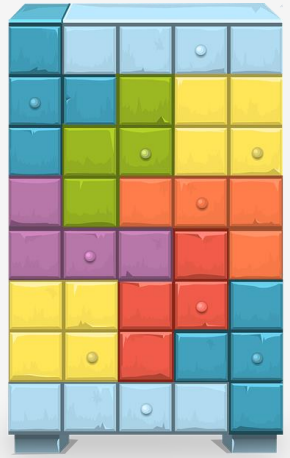
- Kolekcja nie pozwalająca na bezpośredni dostęp do obiektu poprzez podanie np. indeksu
- Nie pozwala przechowywać duplikatów
- Aby dostać się do obiektu musimy skorzystać z pętli foreach lub specjalnego typu Iterator



Kolekcje

Set– metody interfejsu Set<E>:

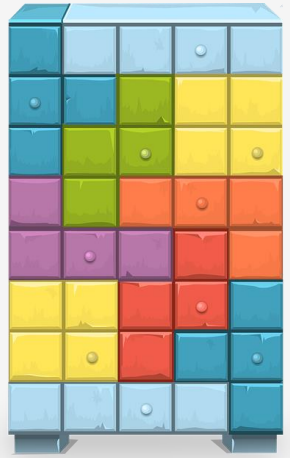
- *add(E e) – dodaje element do zbioru*
- *addAll(Collection) – dodaje kolekcję do zbioru*
- *contains(E e) – zwraca true jeśli element znajduje się w zbiorze*
- *isEmpty() – zwraca true jeśli zbiór jest pusty*
- *iterator() – zwraca obiekt typu Iterator umożliwiający iterowanie zbioru*
- *remove (E e) – usuwa wybrany element*
- *size() – zwraca rozmiar listy*



Kolekcje

Set – implementacje interfejsu:

- HashSet
  - Najczęściej występująca – dobra wydajnościowo
  - Brak zachowania kolejności
- LinkedHashSet
  - Zachowuje kolejność wpisywanych elementów (jak w Liście)
- TreeSet
  - Umieszcza nowe elementy kolekcji poprzez użycie Comparatora
  - Wszystkie elementy od razu posortowane i zachowują swoją kolejność



Kolekcje – zadanie 2

1. *Utwórz klasę abstrakcyjną Figura*
2. *Dodaj metodę abstrakcyjną obliczPole():double*
3. *Zaimplementuj interfejs Comparable tak aby sortować względem wielkości pola*
4. *Nadpisz metodę .toString() aby zwracać wielkość pola*
5. *Napisz klasy Kwadrat oraz Prostokąt dziedziczące po Figurze*
6. *Utwórz kilka obiektów typu Kwadrat i Prostokąt i umieść w Secie przechowującym typ Figura*
7. *Wydrukuj wszystkie obiekty*
8. *Podmień implementację seta i zaobserwuj różnice*
    1. *HashSet*
    2. *LinkedHashSet*
    3. *TreeSet*

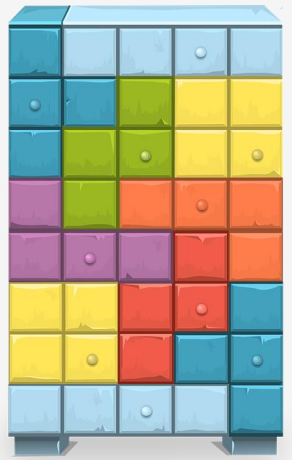
Kolekcje – zadanie 3

1. *Napisz program do losowania gry w lotto przy pomocy Seta*
    1. *Utwórz Klasę LottoGra*
    2. *Dodaj metodę zagraj():void, która najpierw odpyta użytkownika o 6 liczb w zakresie 1-49 i zapisze wynik w postaci Setu*
    3. *Następnie wywoła metodę prywatną przeprowadzLosowanie():Set<Integer>, która zwróci 6 losowo wybranych liczb*
    4. *Ostatecznie prześle oba sety do metody zwrocWynik(Set, Set):int która przekaże ilość trafionych liczb*
    5. *Na koniec wydrukuje wiadomość podsumowującą tj. jakie liczby obstawił użytkownik a jakie wygenerował kompu oraz liczbę trafień użytkownika*

Kolekcje

Mapa:

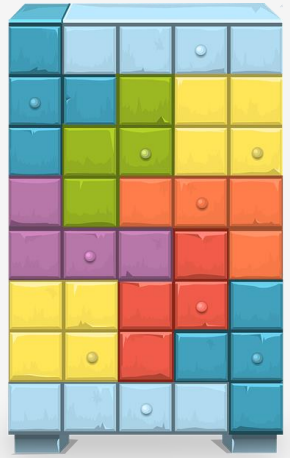
- Stanowi zbiór klucz→ wartość
- Klucze są unikatowe
- Wartości mogą się powtarzać
- Dostęp do wartości odbywa się poprzez podanie klucza



Kolekcje

Map – metody interfejsu Map<K,V>:

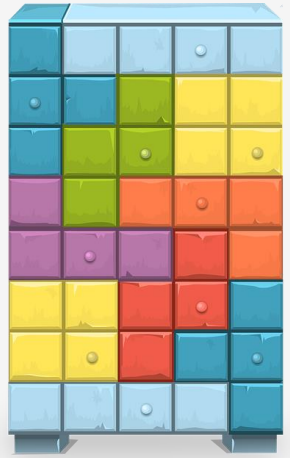
- *containsKey(Object o) – zwraca true jeśli mapa zawiera wskazany klucz*
- *containsValue(Object o) – zwraca true jeśli mapa zawiera wskazaną wartość*
- *get(Object key) – zwraca wartość dla wskazanego klucza*
- *isEmpty() – zwraca true jeśli zbiór jest pusty*
- *keySet() – zwraca set wszystkich kluczy mapie*
- *put(Key ket, Value value) – umieszcza nową parę klucz-wartość w mapie*
- *remove(Object key) – usuwa wybraną parę klucz-wartość na podstawie klucza*
- *replace (Key key, Value value) – podmienia wartość dla wskazanego klucza*
- *size() – zwraca rozmiar listy*



Kolekcje

Map – implementacje interfejsu:

- HashMap
  - Najczęściej występująca – dobra wydajnościowo
  - Brak zachowania kolejności
- LinkedHashMap
  - Zachowuje kolejność wpisywanych elementów (jak w Liście)
- TreeMap
  - Umieszcza nowe elementy kolekcji poprzez użycie Comparatora
  - Wszystkie elementy od razu posortowane względem klucza i zachowują swoją kolejność



Kolekcje – zadanie 4

1. *Utwórz klasę MapaTest a w niej metodę psvm*
2. *Utwórz hashmapę gdzie kluczem będzie String – imię a wartościa int – wiek*
3. *Dodaj do mapy kilka wystąpień*
4. *Wyświetl mapę (sout)*
5. *Spróbuj dodać do mapy obecny już klucz z inna wartością – co się stanie?*
6. *Przeiteruj mapę za pomocą pętli for (.keySet())*
7. *Sprawdź zachowanie dla innych implementacji mapy:*
    1. *LinkedHashMap*
    2. *TreeMap*

Kolekcje – zadanie 5

1. *Napisz program do zliczania wystąpień słów w tekście, w tym celu:*
2. *Dodaj metodę zliczWystapieniaSlow(String tekst):Map<String, Integer>*
    1. *Metoda pobiera tekst*
    2. *Następnie rozdziela go na wystąpienia słów*
    3. *Tworzy mapę ‚słowo' → ilość wystąpień*
    4. *Iteruje po wszystkich słowach w zadanym tekscie*
    5. *Dla każdego słowa wyciąga ilość zliczonych słów z mapy i dorzuca kolejne wystąpienie*
3. *Program wyświetla wszystkie odkryte słowa wraz z ich liczebnością*
4. ** Program wyświetla wszystkie odkryte słowa wraz z ich liczebnością w kolejności od najczęściej występującego do najrzadziej występującego*

Kolekcje – zadanie 6

1. *Napisz program do tworzenia skorowidzu liter*
2. *Utwórz metode skorowidzLiterowy(String tekst):Map<String, Set<Integer>>*
    1. *Metoda rozdziela zadany tekst na pojedyncze litery*
    2. *Następnie iteruje od 0 do n pojedynczych liter*
    3. *Aktualizuje indeksy wystąpień dla każdej litery*
    4. *Zwraca mapę w postaci :*
        1. *litera -> [indeksy wystąpień]*
3. *Wyświetla wynik w postaci :*
    1. *[a -> [2, 3, 5], b- > [1,4]]*
4. *Np.*
    1. *Dla tekstu „Hello”: [e-> [1], l -> [2, 3], o -> [4] , H->[0]]*

Bibliografia

1. https://pixabay.com/pl/młotek-narzędzia-metalowe-celuj-w-33617/ (dostęp 28.12.2017)
2. *https://pixabay.com/pl/bunnies-królików-przedsiębiorstwo-151390/ (dostęp 28.12.2017)*
3. *http://www.baeldung.com/java-varargs (dostęp 28.12.2017)*
4. https://pixabay.com/pl/niebezpieczeństwo-panelu-uwagi-3061159/ (dostęp 29.12.2017)
5. https://docs.oracle.com/javase/tutorial/essential/exceptions/catchOrDeclare.html (dostęp 29.12.2017)
6. https://pixabay.com/pl/pokemon-pokeball-pokemon-idź-1536849/ (dostęp 30.12.2017)
7. https://pixabay.com/pl/banderą-formuła-chequered-speedway-42581/ (dostęp 30.12.2017)
8. https://pl.wikipedia.org/wiki/Wyrażenie_regularne (dostęp 30.12.2017)
9. https://pl.wikipedia.org/wiki/Interfejs_(programowanie_obiektowe) (dostęp 30.12.2017)
10. https://pixabay.com/pl/symbol-usb-komputery-symbol-1906474/ (dostęp 30.12.2017)
11. https://pixabay.com/pl/człowiek-głowy-twarz-awatar-157699/ (dostęp 31.12.2017)
12. https://pl.wikipedia.org/wiki/Programowanie_uogólnione (dostęp 31.12.2017)
13. https://pixabay.com/pl/szafki-na-ubrania-półka-szafki-575373/ (02.01.2018)
14. https://docs.oracle.com/javase/tutorial/collections/interfaces/list.html (dostęp 02.01.2018)
15. https://www.javatpoint.com/difference-between-arraylist-and-linkedlist (dostęp 02.01.2018)