

Stefan Sokołowski

PROGRAMOWANIE OBIEKTOWE II

wykład 6

**Inst. Informatyki Stosowanej, ANS
Elbląg, 2024/2025**

Filtrowanie list



Filtrowanie list

Porownanie — od par do wartości logicznych

Wlasnosc — od pojedynczych elementów do wartości logicznych



Filtrowanie list

OGÓLNIEJSZE

Porównanie — od par do wartości logicznych

→ Własność — od pojedynczych elementów do wartości logicznych



Filtrowanie list

OGÓLNIEJSZE

Porównanie — od par do wartości logicznych

→ Własność — od pojedynczych elementów do wartości logicznych



Własnościami można na przykład filtrować listy

Filtrowanie list

OGÓLNIJSZE

Porównanie — od par do wartości logicznych

→ Własność — od pojedynczych elementów do wartości logicznych



Własnościami można na przykład **filtrować** listy:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Filtrowanie list

OGÓLNIJSZE

Porównanie — od par do wartości logicznych

→ Własność — od pojedynczych elementów do wartości logicznych



Własnościami można na przykład filtrować listy:

[0, ■, 2, ■, 4, ■, 6, ■, 8, ■]

parzyste

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Filtrowanie list

OGÓLNIJSZE

Porównanie — od par do wartości logicznych

→ Własność — od pojedynczych elementów do wartości logicznych



Własnościami można na przykład filtrować listy:

[0, ■, 2, ■, 4, ■, 6, ■, 8, ■]

parzyste

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

niepodzielne przez 3

[■, 1, 2, ■, 4, 5, ■, 7, 8, ■]

Filtrowanie list

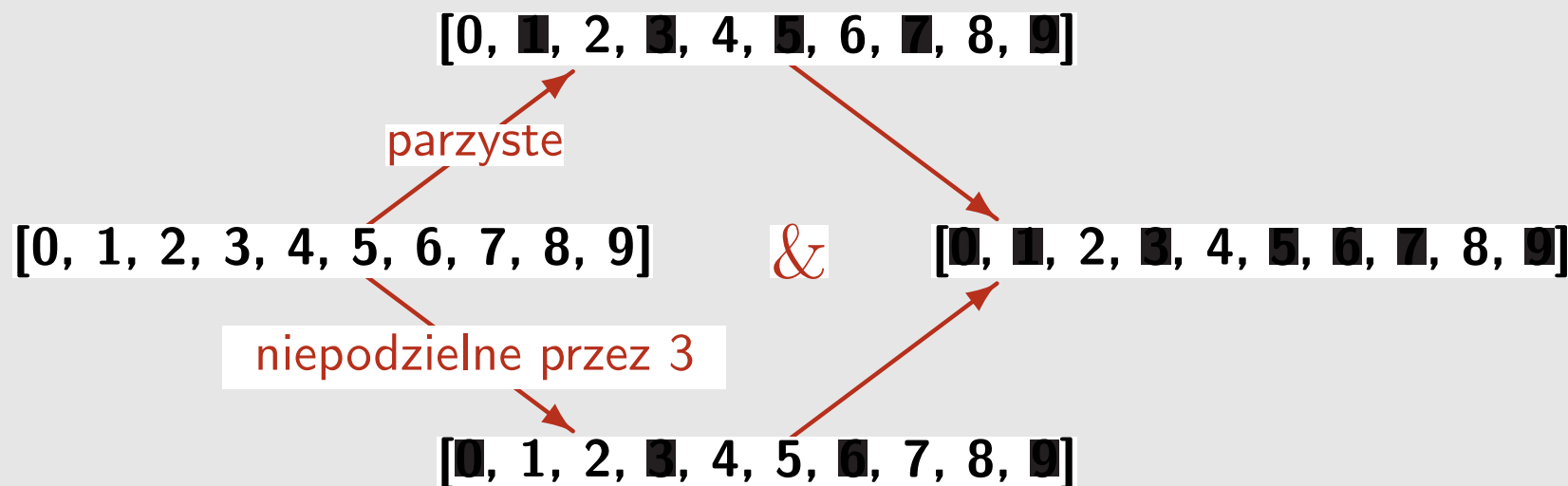
OGÓLNIJSZE

Porównanie — od par do wartości logicznych

→ Własność — od pojedynczych elementów do wartości logicznych



Własnościami można na przykład filtrować listy:

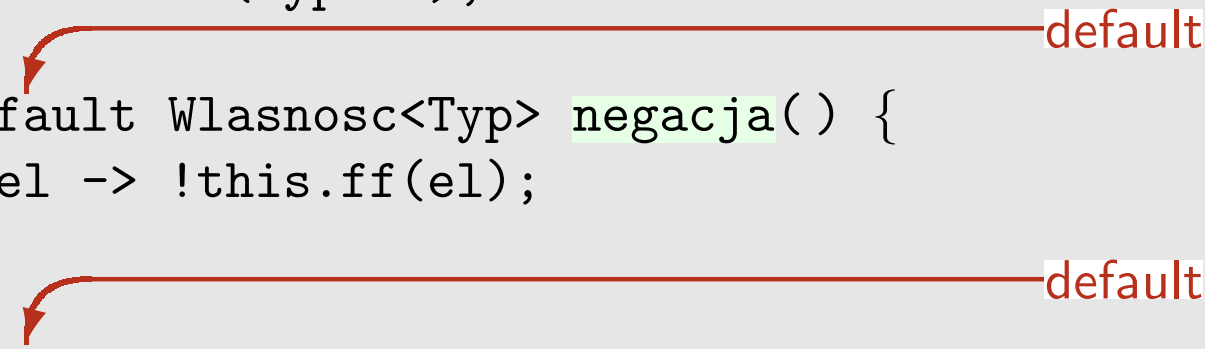


Filtrowanie list

```
interface Wlasnosc<Typ> {  
    public Boolean ff(Typ el);  
  
    public default Wlasnosc<Typ> negacja() {  
        return el -> !this.ff(el);  
    }  
  
    public default Wlasnosc<Typ> koniunkcja(Wlasnosc w1) {  
        return el -> this.ff(el) && w1.ff(el);  
    }  
}
```

Filtrowanie list

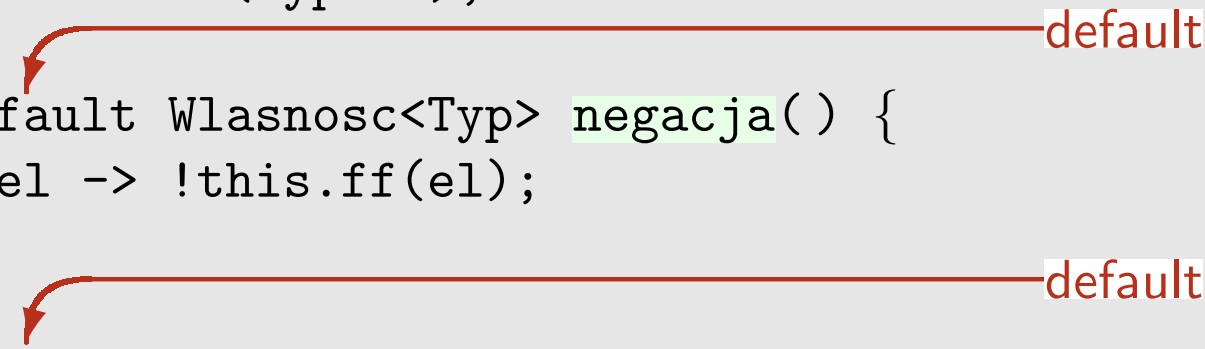
```
interface Wlasnosc<Typ> {  
    public Boolean ff(Typ el);  
  
    public default Wlasnosc<Typ> negacja() {  
        return el -> !this.ff(el);  
    }  
  
    public default Wlasnosc<Typ> koniunkcja(Wlasnosc w1) {  
        return el -> this.ff(el) && w1.ff(el);  
    }  
}
```



The diagram consists of two red arrows pointing from the word 'default' to the 'default' keyword in the method signatures. The first arrow points from the 'default' label to the 'default' keyword in the 'negacja()' method. The second arrow points from the 'default' label to the 'default' keyword in the 'koniunkcja()' method.

Filtrowanie list

```
interface Wlasnosc<Typ> {  
    public Boolean ff(Typ el);  
  
    public default Wlasnosc<Typ> negacja() {  
        return el -> !this.ff(el);  
    }  
  
    public default Wlasnosc<Typ> koniunkcja(Wlasnosc w1) {  
        return el -> this.ff(el) && w1.ff(el);  
    }  
}
```



W nowszych wersjach Jawy (począwszy od Java 8) interfejs może zawierać metody **domyślne**. Klasy implementujące nie muszą (choć mogą) je definiować (przedefiniowywać).

Filtrowanie list

```
class Lista<Typ> extends ArrayList<Typ> {  
    public Lista() { super(); }  
    public Lista(Typ[] a) { super(Arrays.asList(a)); }  
  
    public Lista filter(Wlasnosc w1) {  
        Lista ls = new Lista();  
        for (Typ el : this)  
            if (w1.ff(el)) ls.add(el);  
        return ls;  
    }  
}
```

Filtrowanie list

```
class Lista<Typ> extends ArrayList<Typ> {  
    public Lista() { super(); }  
    public Lista(Typ[] a) { super(Arrays.asList(a)); }  
  
    public Lista filter(Wlasnosc w1) {  
        Lista ls = new Lista();  
        for (Typ el : this)  
            if (w1.ff(el)) ls.add(el);  
        return ls;  
    }  
}
```

konstruktory

Filtrowanie list

```
class Lista<Typ> extends ArrayList<Typ> {  
    public Lista() { super(); }  
    public Lista(Typ[] a) { super(Arrays.asList(a)); }  
  
    public Lista filter(Wlasnosc w1) {  
        Lista ls = new Lista();  
        for (Typ el : this)  
            if (w1.ff(el)) ls.add(el);  
        return ls;  
    }  
}
```

konstruktory

dołączanie elementów spełniających w1.ff

Filtrowanie list

```
class Lista<Typ> extends ArrayList<Typ> {  
    public Lista() { super(); }  
    public Lista(Typ[] a) { super(Arrays.asList(a)); }  
  
    public Lista filter(Wlasnosc wl) {  
        Lista ls = new Lista();  
        for (Typ el : this)  
            if (wl.ff(el)) ls.add(el);  
        return ls;  
    }  
}
```

konstruktory

dołączanie elementów spełniających wl.ff

iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog06/P1_Filtry.java

Filtrowanie list

Z własnościami nie musimy sami się męczyć, bo...

Java zawiera interface `Predicate<Typ>`
analogiczny do powyższego `Wlasnosc<Typ>`.

Filtrowanie list

Z własnościami nie musimy sami się męczyć, bo...

Java zawiera interface `Predicate<Typ>`
analogiczny do powyższego `Wlasnosc<Typ>`.

Klasa `ArrayList<Typ>` zawiera filtrowanie listy przez predykat

Filtrowanie list

Z własnościami nie musimy sami się męczyć, bo...

Java zawiera interface `Predicate<Typ>`
analogiczny do powyższego `Wlasnosc<Typ>`.

Klasa `ArrayList<Typ>` zawiera filtrowanie listy przez predykat:
metoda `removeIf` działa dualnie do powyższej `filter`

Filtrowanie list

Z własnościami nie musimy sami się męczyć, bo...

Java zawiera interface `Predicate<Typ>`
analogiczny do powyższego `Wlasnosc<Typ>`.

Klasa `ArrayList<Typ>` zawiera filtrowanie listy przez predykat:
metoda `removeIf` działa dualnie do powyższej `filter` — włącza do listy
wynikowej elementy **niespełniające** podanego predykatu

Filtrowanie list

Z własnościami nie musimy sami się męczyć, bo...

Java zawiera interface `Predicate<Typ>`
analogiczny do powyższego `Wlasnosc<Typ>`.

Klasa `ArrayList<Typ>` zawiera filtrowanie listy przez predykat:
metoda `removeIf` działa dualnie do powyższej `filter` — włącza do listy
wynikowej elementy **niespełniające** podanego predykatu (czyli **spełniające**
usuwa).

Filtrowanie list

Z własnościami nie musimy sami się męczyć, bo...

Java zawiera interface `Predicate<Typ>`
analogiczny do powyższego `Wlasnosc<Typ>`.

Klasa `ArrayList<Typ>` zawiera filtrowanie listy przez predykat:
metoda `removeIf` działa dualnie do powyższej `filter` — włącza do listy
wynikowej elementy **niespełniające** podanego predykatu (czyli **spełniające**
usuwa).

iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog06/P2_Filtry.java

Filtrowanie list

Z własnościami nie musimy sami się męczyć, bo...

Java zawiera interface `Predicate<Typ>`
analogiczny do powyższego `Wlasnosc<Typ>`.

Klasa `ArrayList<Typ>` zawiera filtrowanie listy przez predykat:
metoda `removeIf` działa dualnie do powyższej `filter` — włącza do listy
wynikowej elementy **niespełniające** podanego predykatu (czyli **spełniające**
usuwa).

iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog06/P2_Filtry.java

Proszę przeglądać dokumentację!

W każdej nowej klasie...

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt1 = new Punkt(2,3);  
        Punkt pt2 = new Punkt(2,3);  
        System.out.println(pt1 == pt2);  
        System.out.println(pt1.equals(pt2));  
    }  
}
```

W każdej nowej klasie...

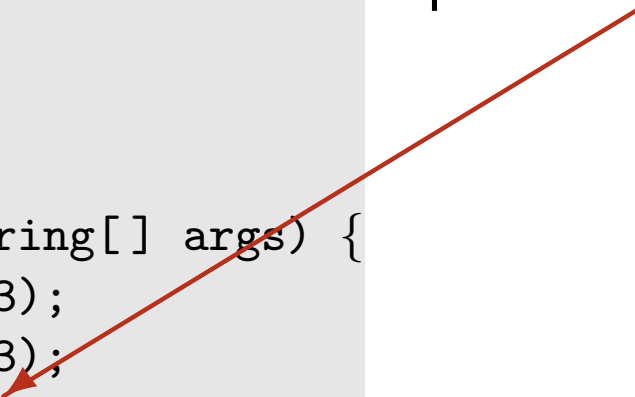
```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt1 = new Punkt(2,3);  
        Punkt pt2 = new Punkt(2,3);  
        System.out.println(pt1 == pt2);  
        System.out.println(pt1.equals(pt2));  
    }  
}
```

Program tworzy dwa
identyczne punkty i sprawdza
ich równość

W każdej nowej klasie...

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt1 = new Punkt(2,3);  
        Punkt pt2 = new Punkt(2,3);  
        System.out.println(pt1 == pt2);  
        System.out.println(pt1.equals(pt2));  
    }  
}
```

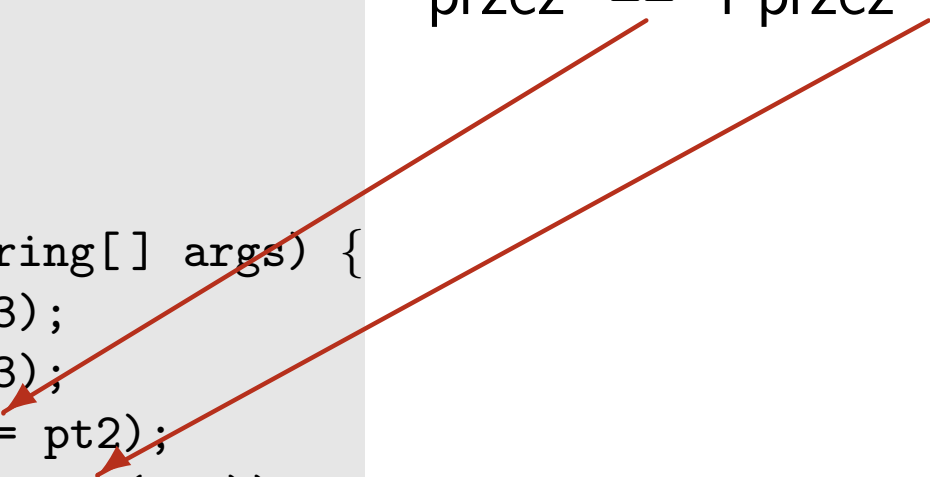
Program tworzy dwa
identyczne punkty i sprawdza
ich równość
przez ==



W każdej nowej klasie...

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt1 = new Punkt(2,3);  
        Punkt pt2 = new Punkt(2,3);  
        System.out.println(pt1 == pt2);  
        System.out.println(pt1.equals(pt2));  
    }  
}
```

Program tworzy dwa
identyczne punkty i sprawdza
ich równość
przez `==` i przez `equals()`.



W każdej nowej klasie...

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt1 = new Punkt(2,3);  
        Punkt pt2 = new Punkt(2,3);  
        System.out.println(pt1 == pt2);  
        System.out.println(pt1.equals(pt2));  
    }  
}
```

Program tworzy dwa
identyczne punkty i sprawdza
ich równość
przez `==` i przez `equals()`.

Wydruk:

false
false

W każdej nowej klasie...

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt1 = new Punkt(2,3);  
        Punkt pt2 = new Punkt(2,3);  
        System.out.println(pt1 == pt2);  
        System.out.println(pt1.equals(pt2));  
    }  
}
```

Program tworzy dwa
identyczne punkty i sprawdza
ich równość
przez `==` i przez `equals()`.

Wydruk: `false`
`false`

Wniosek:
Dla Jawy dwa osobno
stworzone obiekty **nie są**
równe.

W każdej nowej klasie...

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt1 = new Punkt(2,3);  
        Punkt pt2 = new Punkt(2,3);  
        System.out.println(pt1 == pt2);  
        System.out.println(pt1.equals(pt2));  
    }  
}
```

Program tworzy dwa **identyczne** punkty i sprawdza ich równość przez `==` i przez `equals()`.

Wydruk: `false`
`false`

Wniosek:
Dla Jawy dwa osobno stworzone obiekty **nie są równe**.

We własnej klasie potrzebna jest własna metoda `equals()`.

W każdej nowej klasie...

Każde dwa obiekty można porównać operatorem `==` i metodą `equals()`.

W każdej nowej klasie. . .

Każde dwa obiekty można porównać operatorem `==` i metodą `equals()`. Ale porównywane są miejsca w pamięci, gdzie te obiekty „mieszkają”, a nie ich struktura wewnętrzna.

W każdej nowej klasie. . .

Każde dwa obiekty można porównać operatorem `==` i metodą `equals()`. Ale porównywane są **miejsca w pamięci**, gdzie te obiekty „mieszkają”, a nie ich struktura wewnętrzna. Najczęściej warto jest metodę `equals()` **przykryć** własną metodą specyficzną dla danej klasy.

W każdej nowej klasie...

Każde dwa obiekty można porównać operatorem `==` i metodą `equals()`. Ale porównywane są **miejsca w pamięci**, gdzie te obiekty „mieszkają”, a nie ich struktura wewnętrzna. Najczęściej warto jest metodę `equals()` **przykryć** własną metodą specyficzną dla danej klasy.

```
class Punkt {
    private double xx, yy;
    public Punkt(double x, double y) {
        xx = x; yy = y;
    }
    public double x() { return xx; }
    public double y() { return yy; }
    public Boolean equals(Punkt pt) {
        return pt.x()==xx && pt.y()==yy;
    }
}

class Test {
    public static void main(String[] args) {
        Punkt pt1 = new Punkt(2,3);
        Punkt pt2 = new Punkt(2,3);
        System.out.println(pt1 == pt2);
        System.out.println(pt1.equals(pt2));
    }
}
```

W każdej nowej klasie...

Każde dwa obiekty można porównać operatorem `==` i metodą `equals()`. Ale porównywane są **miejsca w pamięci**, gdzie te obiekty „mieszkają”, a nie ich struktura wewnętrzna. Najczęściej warto jest metodę `equals()` **przykryć** własną metodą specyficzną dla danej klasy.

Teraz wydruk:

false

true

```
class Punkt {
    private double xx, yy;
    public Punkt(double x, double y) {
        xx = x; yy = y;
    }
    public double x() { return xx; }
    public double y() { return yy; }
    public Boolean equals(Punkt pt) {
        return pt.x()==xx && pt.y()==yy;
    }
}

class Test {
    public static void main(String[] args) {
        Punkt pt1 = new Punkt(2,3);
        Punkt pt2 = new Punkt(2,3);
        System.out.println(pt1 == pt2);
        System.out.println(pt1.equals(pt2));
    }
}
```

W każdej nowej klasie...

Każde dwa obiekty można porównać operatorem `==` i metodą `equals()`. Ale porównywane są **miejsca w pamięci**, gdzie te obiekty „mieszkają”, a nie ich struktura wewnętrzna. Najczęściej warto jest metodę `equals()` **przykryć** własną metodą specyficzną dla danej klasy.

Teraz wydruk:

false

true

Wniosek:

Operator `==` nadal porównuje „miejsca zamieszkania”, ale metoda `equals()` porównuje wartości.

```
class Punkt {
    private double xx, yy;
    public Punkt(double x, double y) {
        xx = x; yy = y;
    }
    public double x() { return xx; }
    public double y() { return yy; }
    public Boolean equals(Punkt pt) {
        return pt.x()==xx && pt.y()==yy;
    }
}

class Test {
    public static void main(String[] args) {
        Punkt pt1 = new Punkt(2,3);
        Punkt pt2 = new Punkt(2,3);
        System.out.println(pt1 == pt2);
        System.out.println(pt1.equals(pt2));
    }
}
```

W każdej nowej klasie...

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt = new Punkt(2,3);  
        System.out.println(pt);  
    }  
}
```

W każdej nowej klasie...

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt = new Punkt(2,3);  
        System.out.println(pt);  
    }  
}
```

Program tworzy punkt i go drukuje.

W każdej nowej klasie...

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt = new Punkt(2,3);  
        System.out.println(pt);  
    }  
}
```

Program tworzy punkt i go drukuje. Wydruk:

Punkt@512ddf17

W każdej nowej klasie...

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt = new Punkt(2,3);  
        System.out.println(pt);  
    }  
}
```

Program tworzy punkt i go drukuje. Wydruk:

Punkt@512ddf17

Wniosek:

W Javie format wydruku obiektu z własnoręcznie zdefiniowanej klasy jest **nieczytelny**.

W każdej nowej klasie...

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt = new Punkt(2,3);  
        System.out.println(pt);  
    }  
}
```

Program tworzy punkt i go drukuje. Wydruk:

Punkt@512ddf17

Wniosek:

W Javie format wydruku obiektu z własnoręcznie zdefiniowanej klasy jest **nieczytelny**.

We własnej klasie potrzebna jest własna metoda toString().

W każdej nowej klasie. . .

Na każdym obiekcie określona jest metoda `toString()` używana przez drukowanie systemowe; tak więc każdy obiekt da się wydrukować.

W każdej nowej klasie. . .

Na każdym obiekcie określona jest metoda `toString()` używana przez drukowanie systemowe; tak więc każdy obiekt da się wydrukować. Ale ten wydruk jest nieczytelny i prawie zawsze bezużyteczny.

W każdej nowej klasie. . .

Na każdym obiekcie określona jest metoda `toString()` używana przez drukowanie systemowe; tak więc każdy obiekt da się wydrukować. Ale ten wydruk jest nieczytelny i prawie zawsze bezużyteczny. Najczęściej warto jest metodę `toString()` **przykryć** własną metodą specyficzną dla danej klasy.

W każdej nowej klasie...

Na każdym obiekcie określona jest metoda `toString()` używana przez drukowanie systemowe; tak więc każdy obiekt da się wydrukować. Ale ten wydruk jest nieczytelny i prawie zawsze bezużyteczny. Najczęściej warto jest metodę `toString()` **przykryć** własną metodą specyficzną dla danej klasy.

```
class Punkt {  
    private double xx, yy;  
    public Punkt(double x, double y) {  
        xx = x; yy = y;  
    }  
    public String toString() {  
        return "(" + xx + ", " + yy + ")";  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Punkt pt = new Punkt(2,3);  
        System.out.println(pt);  
    }  
}
```

W każdej nowej klasie...

Na każdym obiekcie określona jest metoda `toString()` używana przez drukowanie systemowe; tak więc każdy obiekt da się wydrukować. Ale ten wydruk jest nieczytelny i prawie zawsze bezużyteczny. Najczęściej warto jest metodę `toString()` **przykryć** własną metodą specyficzną dla danej klasy.

Teraz wydruk:

(2.0, 3.0)

```
class Punkt {
    private double xx, yy;
    public Punkt(double x, double y) {
        xx = x; yy = y;
    }
    public String toString() {
        return "(" + xx + ", " + yy + ")";
    }
}

class Test {
    public static void main(String[] args) {
        Punkt pt = new Punkt(2,3);
        System.out.println(pt);
    }
}
```

W każdej nowej klasie. . .

W każdej nowej klasie $\mathcal{K}$ warto

- zdefiniować **własną** metodę

`Boolean equals( $\mathcal{K}$  obiekt)`

W każdej nowej klasie...

W każdej nowej klasie $\mathcal{K}$ warto

- zdefiniować **własną** metodę

`Boolean equals( $\mathcal{K}$  obiekt)`

- zdefiniować **własną** metodę

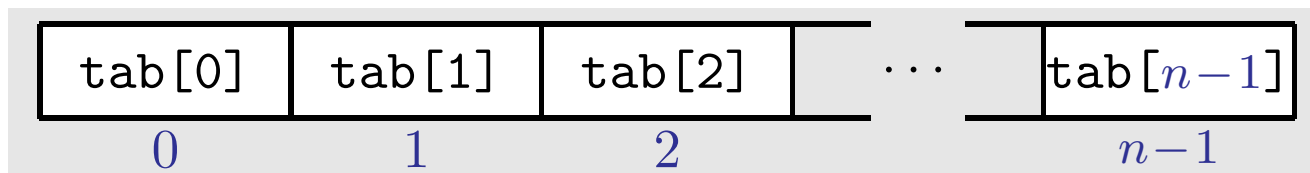
`String toString()`

Tablice

```
Typ[] tab = new Typ[n];
```

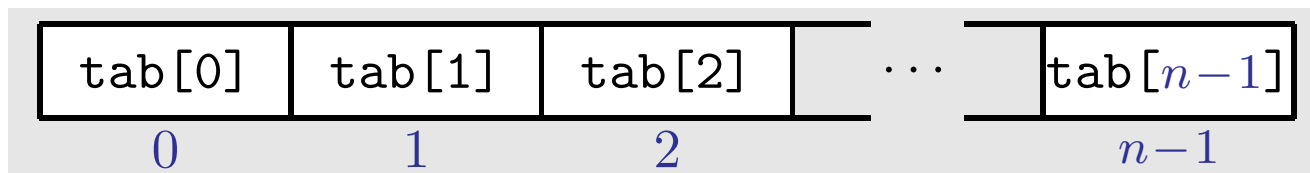
Tablice

```
Typ[] tab = new Typ[n];
```



Tablice

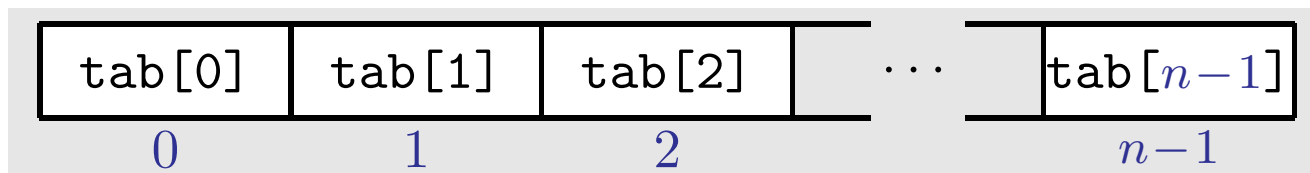
```
Typ[] tab = new Typ[n];
```



$n = \text{tab.length}$ komórek o indeksach od 0 do $n - 1$

Tablice

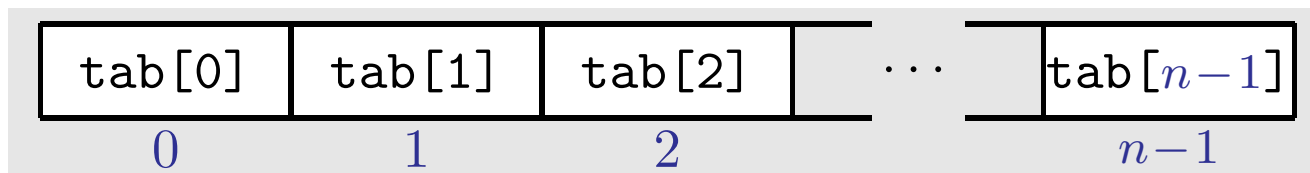
```
Typ[] tab = new Typ[n];
```



$n = \text{tab.length}$ komórek o indeksach od 0 do $n - 1$
do każdej można wpisać obiekt klasy (typu) Typ

Tablice

```
Typ[] tab = new Typ[n];
```



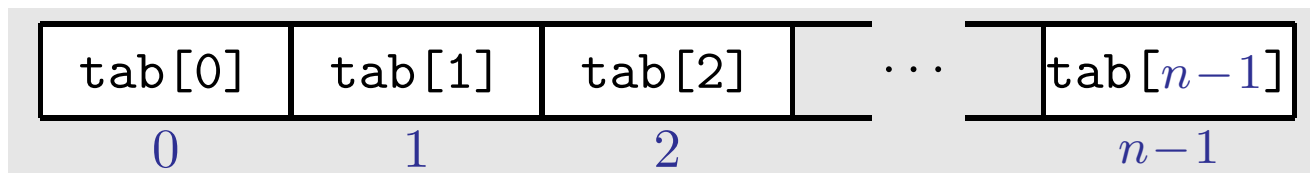
$n = \text{tab.length}$ komórek o indeksach od 0 do $n - 1$
do każdej można wpisać obiekt klasy (typu) Typ

Na przykład:

```
① String[] c = new String[5];    — tablica 5 napisów
```

Tablice

```
Typ[] tab = new Typ[n];
```



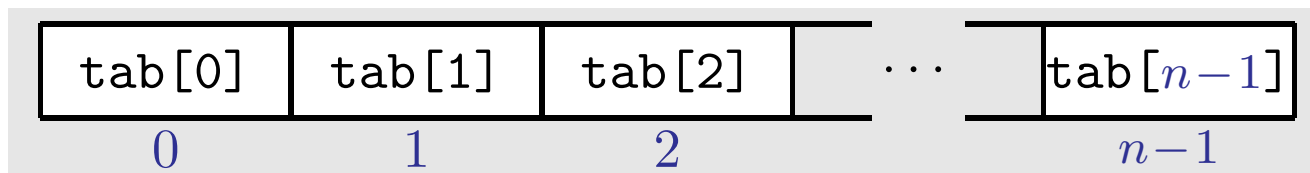
$n = \text{tab.length}$ komórek o indeksach od 0 do $n - 1$
do każdej można wpisać obiekt klasy (typu) Typ

Na przykład:

- ① `String[] c = new String[5];` — tablica 5 napisów
- ① `c = new String[]{"C", "Java"};` — tablica 2 napisów

Tablice

```
Typ[] tab = new Typ[n];
```



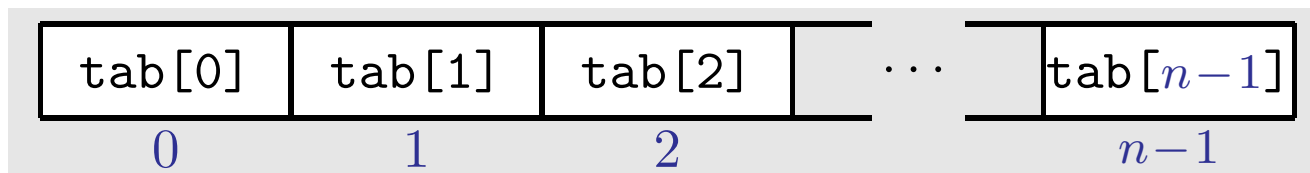
$n = \text{tab.length}$ komórek o indeksach od 0 do $n - 1$
do każdej można wpisać obiekt klasy (typu) Typ

Na przykład:

- ① `String[] c = new String[5];` — tablica 5 napisów
- ① `c = new String[]{"C", "Java"};` — tablica 2 napisów
- ② `c[2] = "Python";`

Tablice

```
Typ[] tab = new Typ[n];
```



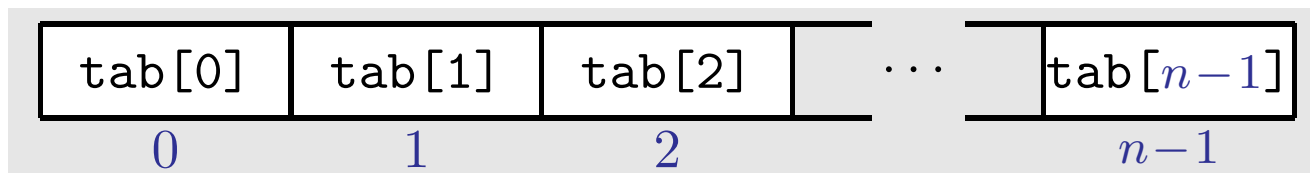
$n = \text{tab.length}$ komórek o indeksach od 0 do $n - 1$
do każdej można wpisać obiekt klasy (typu) Typ

Na przykład:

- ① `String[] c = new String[5];` — tablica 5 napisów
- ① `c = new String[]{"C", "Java"};` — tablica 2 napisów
- ② `c[2] = "Python";` — **BŁĄD**

Tablice

```
Typ[] tab = new Typ[n];
```



$n = \text{tab.length}$ komórek o indeksach od 0 do $n - 1$
do każdej można wpisać obiekt klasy (typu) Typ

Na przykład:

- ① `String[] c = new String[5];` — tablica 5 napisów
- ① `c = new String[]{"C", "Java"};` — tablica 2 napisów
- ② `c[2] = "Python";` — **BŁĄD**

Po ① tablica c już **nie ma** 5 elementów, tylko 2.

Tablica tablic

```
Typ[][] tab = new Typ[n][];
```

Tablica tablic

`Typ[][] tab = new Typ[n][];`

```
int[][] tab = new int[4][];
```

tab:

0

1

2

3

Tablica tablic

`Typ[][] tab = new Typ[n][];`

```
int[][] tab = new int[4][];  
tab[0] = new int[]{ 0, 1, 2};
```



tab:

0	0	1	2
1			
2			
3			

Tablica tablic

`Typ[][] tab = new Typ[n][];`

```
int[][] tab = new int[4][];
```

```
tab[0] = new int[]{ 0, 1, 2};
```

```
tab[1] = new int[]{10, 11};
```

tab:

0	0	1	2
1	10	11	
2			
3			

Tablica tablic

`Typ[][] tab = new Typ[n][];`

```
int[][] tab = new int[4][];
```

```
tab[0] = new int[]{ 0, 1, 2};
```

```
tab[1] = new int[]{10, 11};
```

```
tab[2] = new int[]{20, 21, 22, 23};
```

tab:

0	0	1	2	
1	10	11		
2	20	21	22	23
3				

Tablica tablic

`Typ[][] tab = new Typ[n][];`

```
int[][] tab = new int[4][];
```

```
tab[0] = new int[]{ 0, 1, 2};
```

```
tab[1] = new int[]{10, 11};
```

```
tab[2] = new int[]{20, 21, 22, 23};
```

```
tab[3] = new int[]{};
```

tab:

0	0	1	2	
1	10	11		
2	20	21	22	23
3				

Tablica tablic

`Typ[][] tab = new Typ[n][];`

```
int[][] tab = new int[4][];
```

```
tab[0] = new int[]{ 0, 1, 2};
```

```
tab[1] = new int[]{10, 11};
```

```
tab[2] = new int[]{20, 21, 22, 23};
```

```
tab[3] = new int[]{};
```

tab:

0	0	1	2	
1	10	11		
2	20	21	22	23
3				

Tablica 2-wymiarowa = tablica tablic 1-wymiarowych

Tablica tablic

`Typ[][] tab = new Typ[n][];`

```
int[][] tab = new int[4][];
```

```
tab[0] = new int[]{ 0, 1, 2};
```

```
tab[1] = new int[]{10, 11};
```

```
tab[2] = new int[]{20, 21, 22, 23};
```

```
tab[3] = new int[]{};
```

tab:

0	0	1	2	
1	10	11		
2	20	21	22	23
3				

Tablica 2-wymiarowa = tablica tablic 1-wymiarowych

Jej wiersze **nie muszą** mieć tej samej długości!

Tablica tablic

```
Typ[][] tab = new Typ[n][];
```

```
int[][] tab = new int[4][];
```

```
tab[0] = new int[]{ 0, 1, 2};
```

```
tab[1] = new int[]{10, 11};
```

```
tab[2] = new int[]{20, 21, 22, 23};
```

```
tab[3] = new int[]{};
```

tab:

0	0	1	2	
1	10	11		
2	20	21	22	23
3				

Tablica 2-wymiarowa = tablica tablic 1-wymiarowych

Jej wiersze **nie muszą** mieć tej samej długości!

iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog06/P3-TablicaTablic.java

Tablice

- szybki dostęp do elementu zadanego indeksem

Tablice

- szybki dostęp do elementu zadanego indeksem;
- brak możliwości powiększenia lub zmniejszenia

Tablice

- szybki dostęp do elementu zadanego indeksem;
- brak możliwości powiększenia lub zmniejszenia;
- trudne wstawianie w środek i usuwanie ze środka

Tablice

- szybki dostęp do elementu zadanego indeksem;
- brak możliwości powiększenia lub zmniejszenia;
- trudne wstawianie w środek i usuwanie ze środka (trzeba przesuwać, nie zawsze się da):



Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice
--	----------------

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice
--	----------------

dodać element

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice
dodać element	nie ma można wymienić na inny

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice
dodać element	nie ma można wymienić na inny

usunąć element

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice
dodać element	nie ma można wymienić na inny
usunąć element	nie ma można zaznaczyć nieistotność

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice
dodać element	nie ma można wymienić na inny
usunąć element	nie ma można zaznaczyć nieistotność
sprawdzić należenie	

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice
dodać element	nie ma można wymienić na inny
usunąć element	nie ma można zaznaczyć nieistotność
sprawdzić należenie	trzeba przeszukać

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice
dodać element	nie ma można wymienić na inny
usunąć element	nie ma można zaznaczyć nieistotność
sprawdzić należenie	trzeba przeszukać
moc zbioru (liczba elementów)	

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice
dodać element	nie ma można wymienić na inny
usunąć element	nie ma można zaznaczyć nieistotność
sprawdzić należenie	trzeba przeszukać
moc zbioru (liczba elementów)	zwykle trzeba przeszukać omijając nieistotne

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice
dodać element	nie ma można wymienić na inny
usunąć element	nie ma można zaznaczyć nieistotność
sprawdzić należenie	trzeba przeszukać
moc zbioru (liczba elementów)	<i>zwykle</i> trzeba przeszukać omijając nieistotne

coś zrobić
na wszystkich elementach

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice
dodać element	nie ma można wymienić na inny
usunąć element	nie ma można zaznaczyć nieistotność
sprawdzić należenie	trzeba przeszukać
moc zbioru (liczba elementów)	zwykle trzeba przeszukać omijając nieistotne
coś zrobić na wszystkich elementach	for (Typ elem : tablica) { ... }

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice	interface Collection
dodać element	nie ma można wymienić na inny	
usunąć element	nie ma można zaznaczyć nieistotność	
sprawdzić należenie	trzeba przeszukać	
moc zbioru (liczba elementów)	zwykle trzeba przeszukać omijając nieistotne	
coś zrobić na wszystkich elementach	for (Typ elem : tablica) { ... }	

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice	interface Collection
dodać element	nie ma można wymienić na inny	zbiór. add (element)
usunąć element	nie ma można zaznaczyć nieistotność	
sprawdzić należenie	trzeba przeszukać	
moc zbioru (liczba elementów)	zwykle trzeba przeszukać omijając nieistotne	
coś zrobić na wszystkich elementach	for (Typ elem : tablica) { ... }	

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice	interface Collection
dodać element	nie ma można wymienić na inny	zbiór. add (element)
usunąć element	nie ma można zaznaczyć nieistotność	zbiór. remove (element)
sprawdzić należenie	trzeba przeszukać	
moc zbioru (liczba elementów)	zwykle trzeba przeszukać omijając nieistotne	
coś zrobić na wszystkich elementach	for (Typ elem : tablica) { ... }	

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice	interface Collection
dodać element	nie ma można wymienić na inny	zbiór. add (element)
usunąć element	nie ma można zaznaczyć nieistotność	zbiór. remove (element)
sprawdzić należenie	trzeba przeszukać	zbiór. contains (element)
moc zbioru (liczba elementów)	zwykle trzeba przeszukać omijając nieistotne	
coś zrobić na wszystkich elementach	for (Typ elem : tablica) { ... }	

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice	interface Collection
dodać element	nie ma można wymienić na inny	zbiór. add (element)
usunąć element	nie ma można zaznaczyć nieistotność	zbiór. remove (element)
sprawdzić należenie	trzeba przeszukać	zbiór. contains (element)
moc zbioru (liczba elementów)	zwykle trzeba przeszukać omijając nieistotne	zbiór. size ()
coś zrobić na wszystkich elementach	for (Typ elem : tablica) { ... }	

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice	interface Collection
dodać element	nie ma można wymienić na inny	zbiór. add (element)
usunąć element	nie ma można zaznaczyć nieistotność	zbiór. remove (element)
sprawdzić należenie	trzeba przeszukać	zbiór. contains (element)
moc zbioru (liczba elementów)	zwykle trzeba przeszukać omijając nieistotne	zbiór. size ()
coś zrobić na wszystkich elementach	for (Typ elem : tablica) { ... }	<i>iterator</i> (dalej wyjaśnione)

Kolekcje

Jak w jedno miejsce pozbierać wiele danych tego samego typu?

Tablice nie są najlepszym sposobem realizacji pojęcia **zbioru**...

Operacje na **zbiorach elementów** które mogą być przydatne:

	tablice	interface Collection
dodać element	nie ma można wymienić na inny	zbiór. add (element)
usunąć element	nie ma można zaznaczyć nieistotność	zbiór. remove (element)
sprawdzić należenie	trzeba przeszukać	zbiór. contains (element)
moc zbioru (liczba elementów)	zwykle trzeba przeszukać omijając nieistotne	zbiór. size ()
coś zrobić na wszystkich elementach	for (Typ elem : tablica) { ... }	<i>iterator</i> (dalej wyjaśnione)

Iterator

```
interface Iterator<T> {  
    // czy istnieje jeszcze cos nieprzejrzanego:  
    Boolean hasNext();  
    // jesli istnieje, to go dawaj; inaczej sygnal bledu:  
    T next();  
}
```

Iterator

```
interface Iterator<T> {  
    // czy istnieje jeszcze cos nieprzejrzanego:  
    Boolean hasNext();  
    // jesli istnieje, to go dawaj; inaczej sygnal bledu:  
    T next();  
}
```

użycie iteratora

jawne

niejawne

```
Iterator<T> it = zbiór.iterator();  
while (it.hasNext())  
    akcja na elemencie it.next();
```

Iterator

```
interface Iterator<T> {  
    // czy istnieje jeszcze cos nieprzejrzanego:  
    Boolean hasNext();  
    // jesli istnieje, to go dawaj; inaczej sygnal bledu:  
    T next();  
}
```

użycie iteratora

jawne

```
Iterator<T> it = zbiór.iterator();  
while (it.hasNext())  
    akcja na elemencie it.next();
```

niejawne

```
for (T elem : zbiór)  
    akcja na elemencie elem;
```

Iterator

```
interface Iterator<T> {  
    // czy istnieje jeszcze cos nieprzejrzanego:  
    Boolean hasNext();  
    // jesli istnieje, to go dawaj; inaczej sygnal bledu:  
    T next();  
}
```

użycie iteratora

jawne

```
Iterator<T> it = zbiór.iterator();  
while (it.hasNext())  
    akcja na elemencie it.next();
```

niejawne

```
for (T elem : zbiór)  
    akcja na elemencie elem;
```

To działa dla **wszystkich** klas implementujących **Collection**.

iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog06/P5-Iteracja.java

Iterator

```
interface Iterator<T> {
    // czy istnieje jeszcze cos nieprzejrzanego:
    Boolean hasNext();
    // jesli istnieje, to go dawaj; inaczej sygnal bledu:
    T next();
}
```

użycie iteratora

jawne

```
Iterator<T> it = zbiór.iterator();
while (it.hasNext())
    akcja na elemencie it.next();
```

niejawne

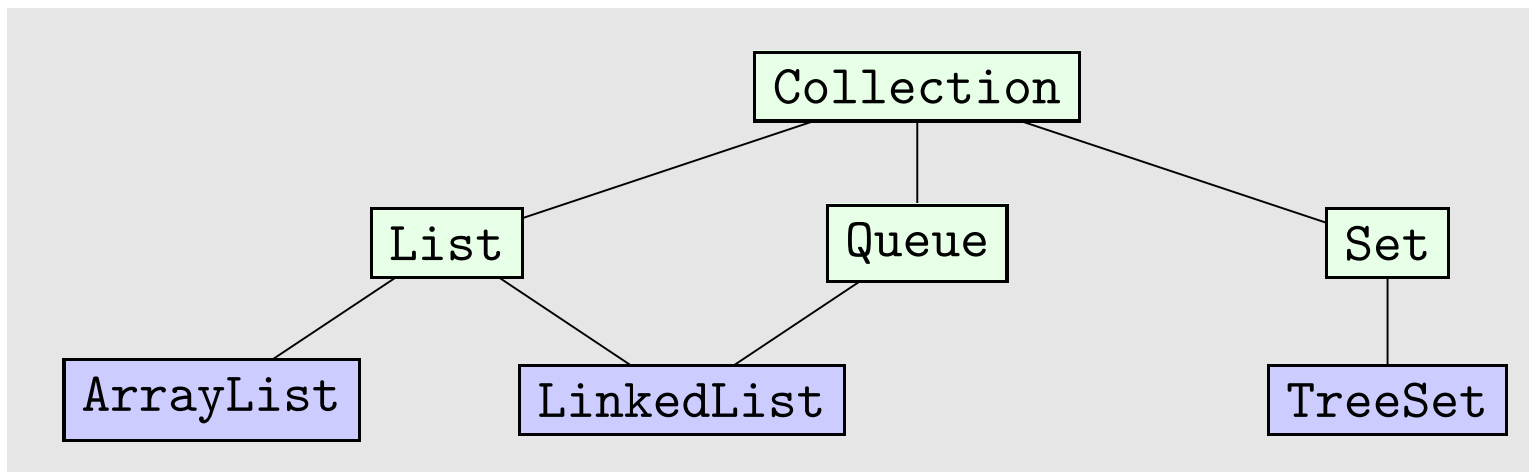
```
for (T elem : zbiór)
    akcja na elemencie elem;
```

To działa dla **wszystkich** klas implementujących `Collection`.

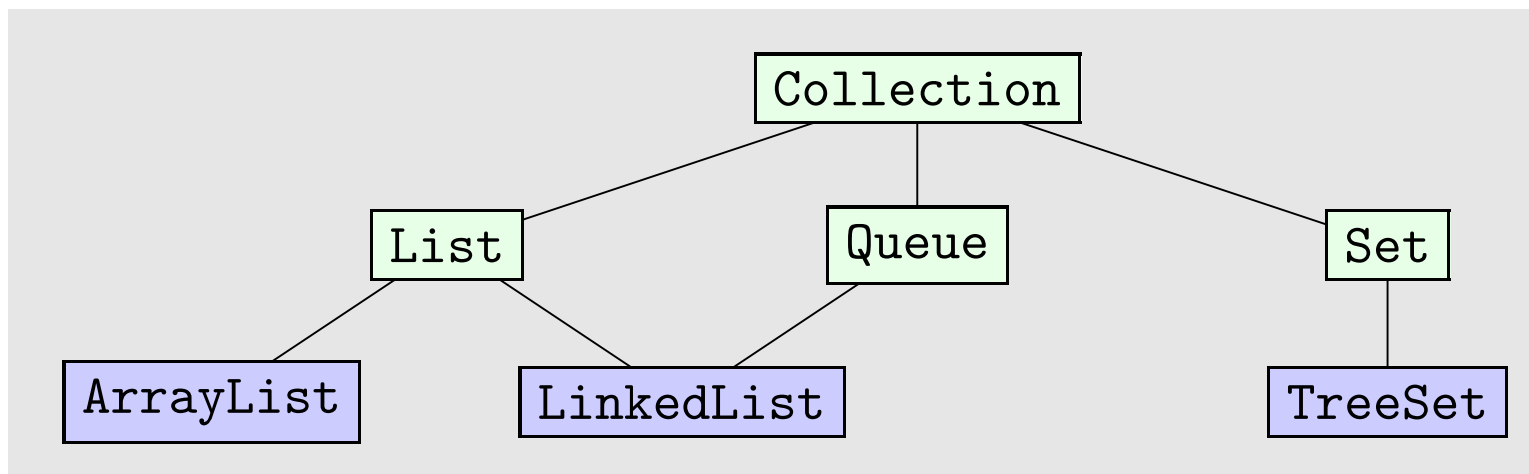
iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog06/P4_Iteracja.java

Czyli dla jakich klas?

Klasy implementujące Collection (przykłady)

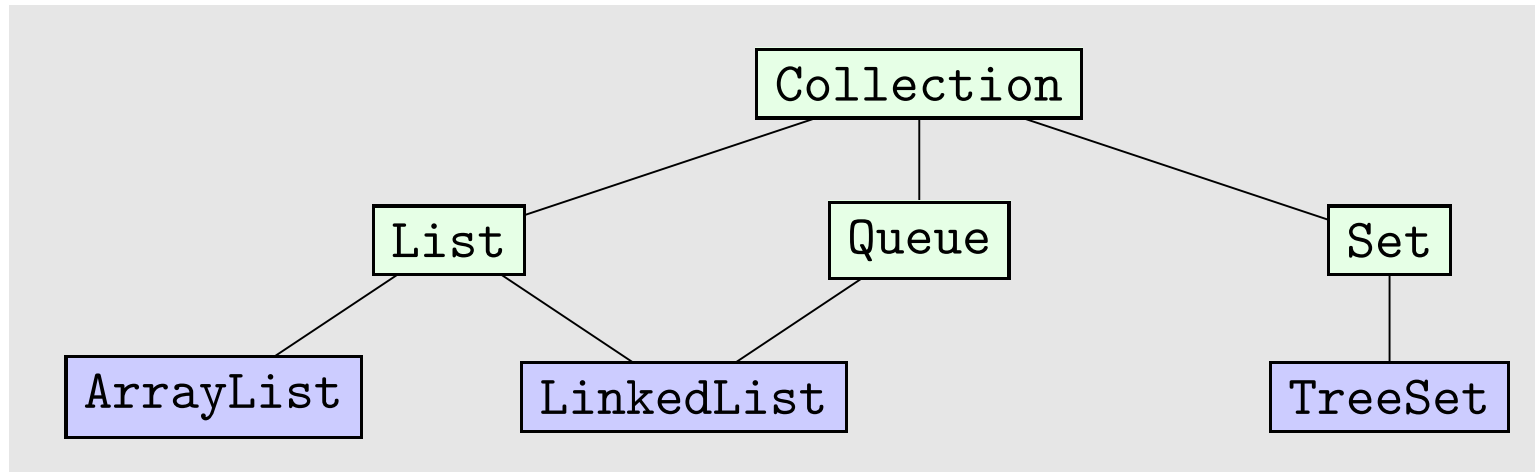


Klasy implementujące Collection (przykłady)



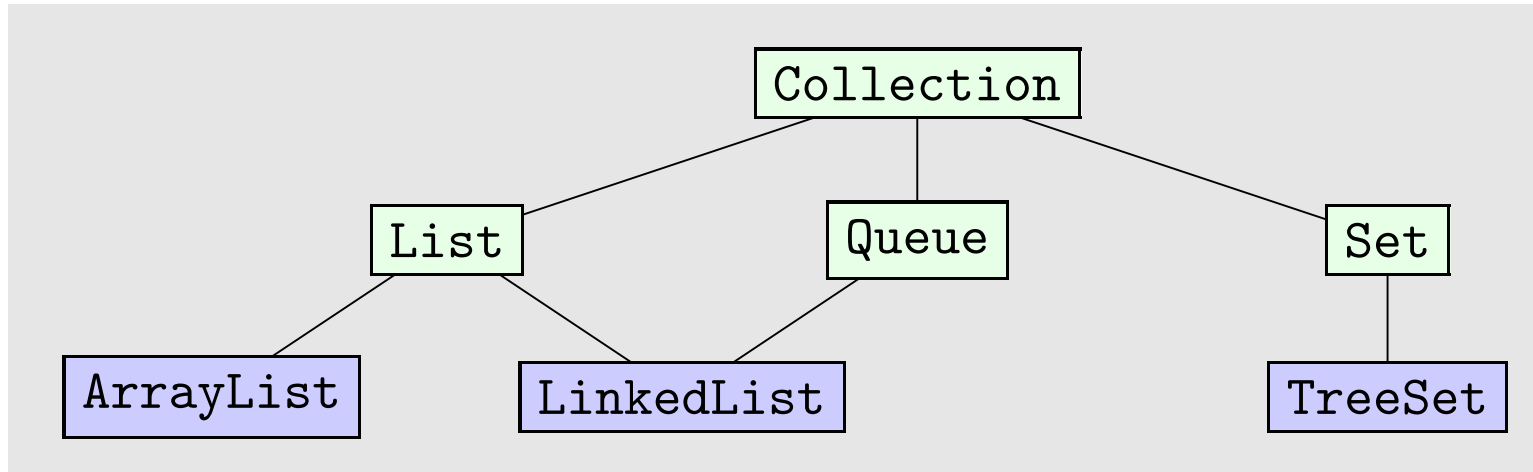
- **ArrayList** — jak tablice, ale automat. zmiana rozmiaru, wstawianie i usuwanie

Klasy implementujące `Collection` (przykłady)



- `ArrayList` — jak tablice, ale automat. zmiana rozmiaru, wstawianie i usuwanie;
Uwaga: `ls[i]` nie działa, trzeba stosować `ls.get(i)` oraz `ls.set(i,x)`

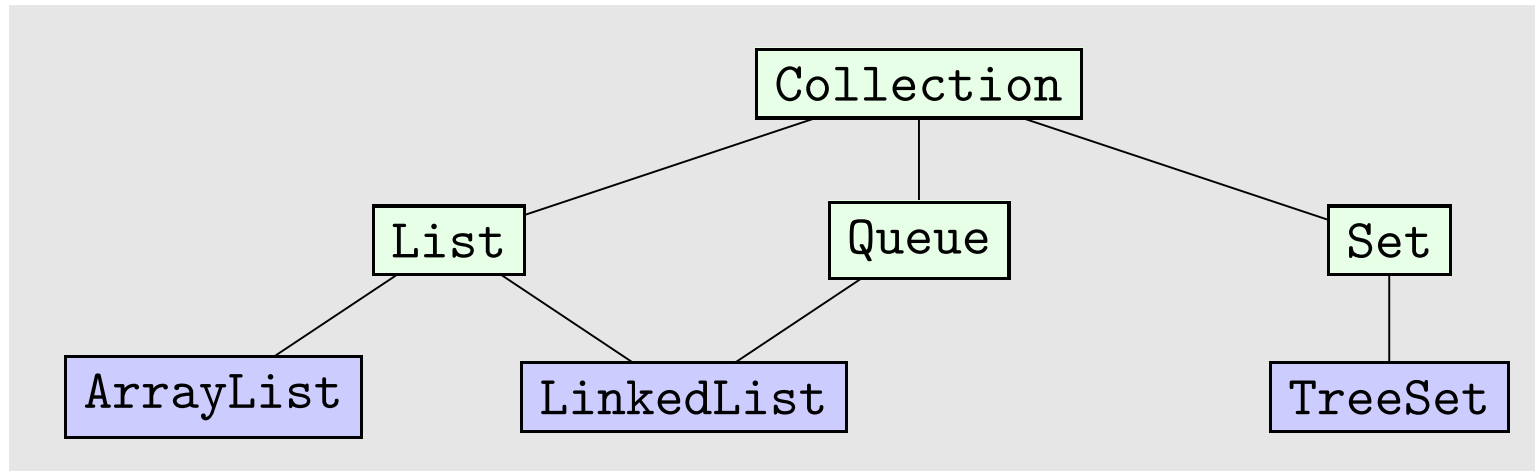
Klasy implementujące Collection (przykłady)



- `ArrayList` — jak tablice, ale automat. zmiana rozmiaru, wstawianie i usuwanie;
Uwaga: `ls[i]` nie działa, trzeba stosować `ls.get(i)` oraz `ls.set(i,x)` ;

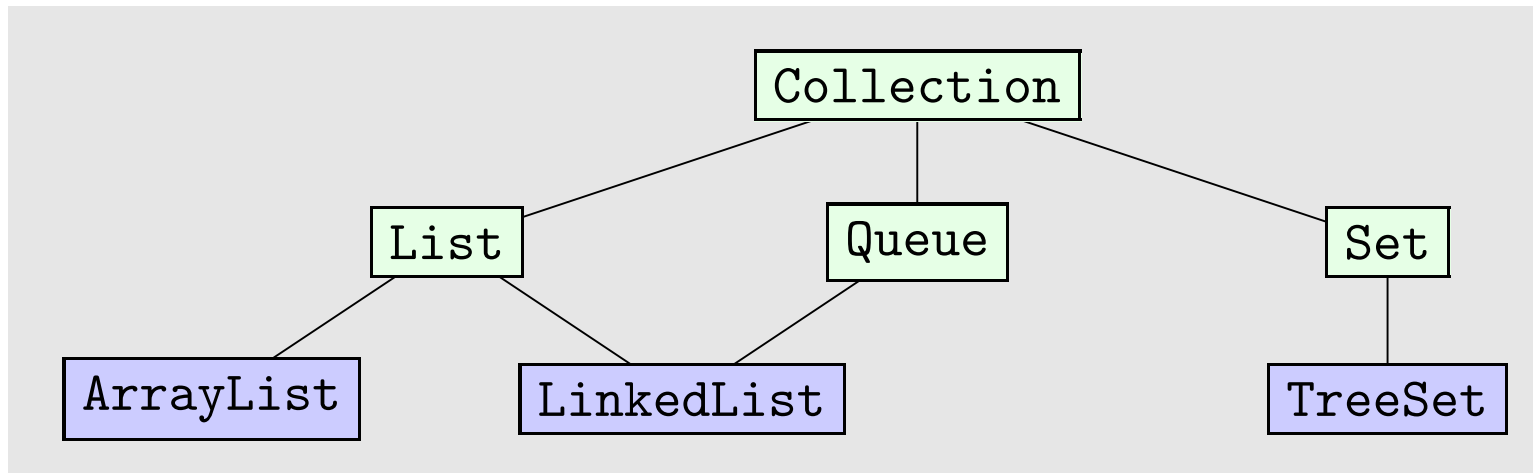
- `LinkedList` — 

Klasy implementujące Collection (przykłady)



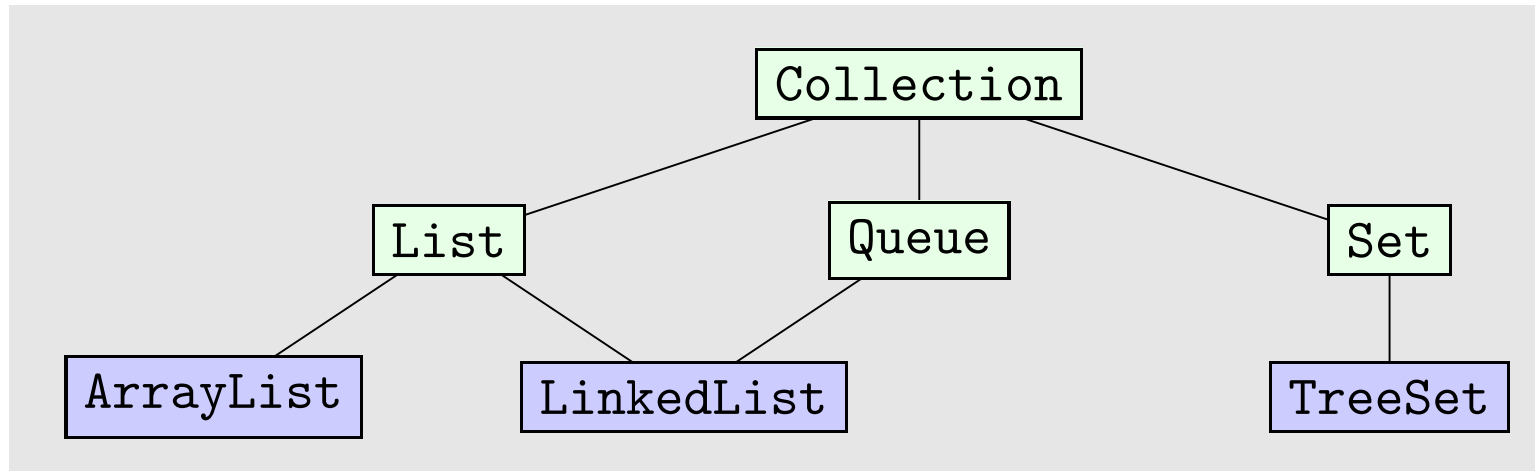
- `ArrayList` — jak tablice, ale automat. zmiana rozmiaru, wstawianie i usuwanie;
Uwaga: `ls[i]` nie działa, trzeba stosować `ls.get(i)` oraz `ls.set(i,x)` ;
- `LinkedList` — 
 - powolny dostęp do elementu zadanego indeksem (przeszukiwanie listy)

Klasy implementujące Collection (przykłady)



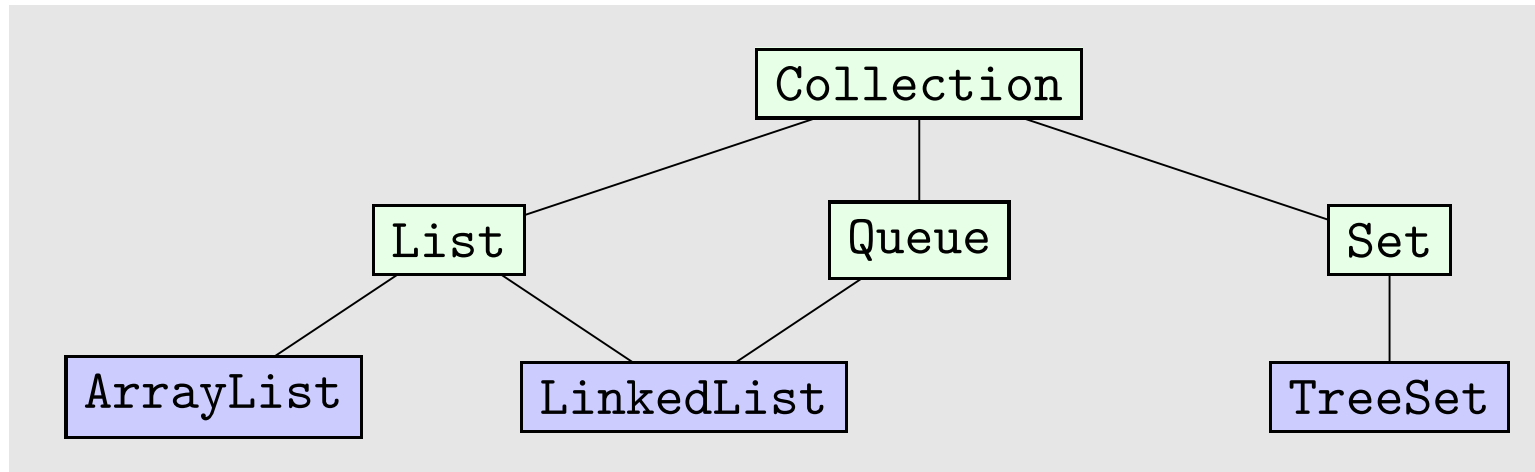
- **ArrayList** — jak tablice, ale automat. zmiana rozmiaru, wstawianie i usuwanie;
Uwaga: `ls[i]` nie działa, trzeba stosować `ls.get(i)` oraz `ls.set(i,x)` ;
- **LinkedList** — 
 - powolny dostęp do elementu zadanego indeksem (przeszukiwanie listy);
 - powiększanie lub zmniejszanie listy bez ograniczeń

Klasy implementujące Collection (przykłady)



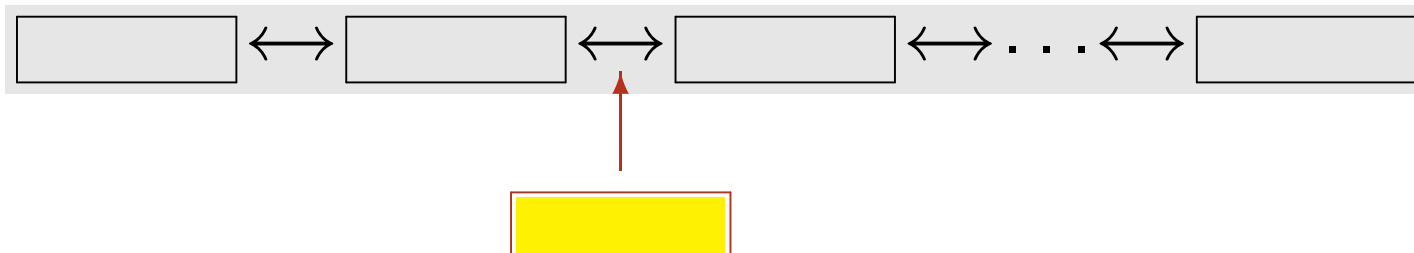
- **ArrayList** — jak tablice, ale automat. zmiana rozmiaru, wstawianie i usuwanie;
Uwaga: `ls[i]` nie działa, trzeba stosować `ls.get(i)` oraz `ls.set(i,x)` ;
- **LinkedList** — 
 - powolny dostęp do elementu zadanego indeksem (przeszukiwanie listy);
 - powiększanie lub zmniejszanie listy bez ograniczeń;
 - łatwe wstawianie w środek i usuwanie ze środka

Klasy implementujące Collection (przykłady)

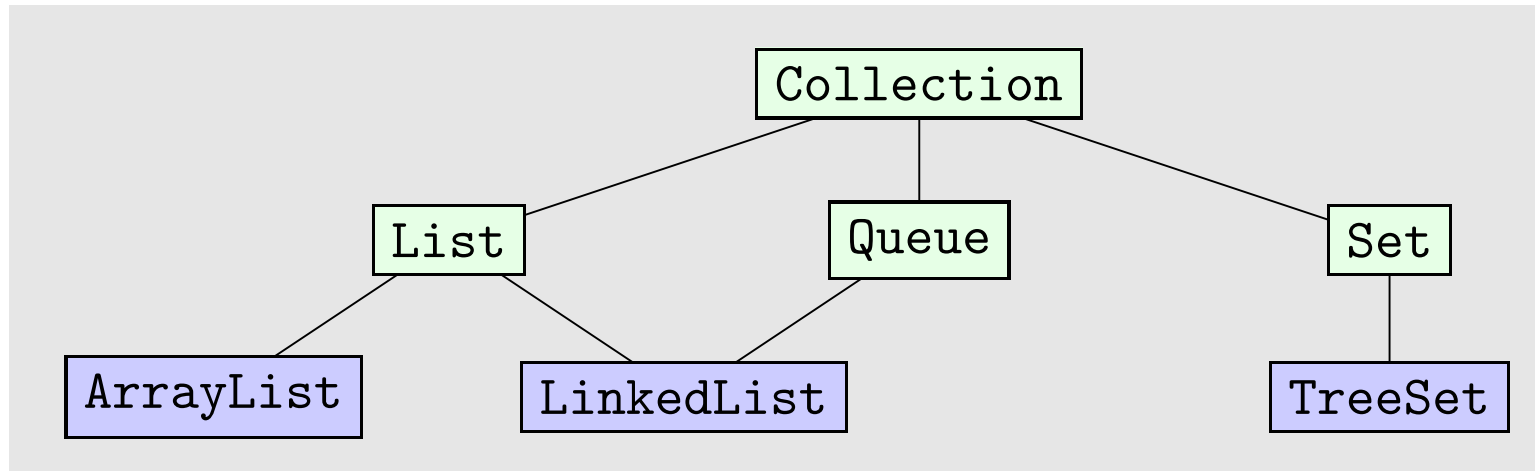


- **ArrayList** — jak tablice, ale automat. zmiana rozmiaru, wstawianie i usuwanie;
Uwaga: `ls[i]` nie działa, trzeba stosować `ls.get(i)` oraz `ls.set(i,x)` ;

- **LinkedList** — 
 - powolny dostęp do elementu zadanego indeksem (przeszukiwanie listy);
 - powiększanie lub zmniejszanie listy bez ograniczeń;
 - łatwe wstawianie w środek i usuwanie ze środka:

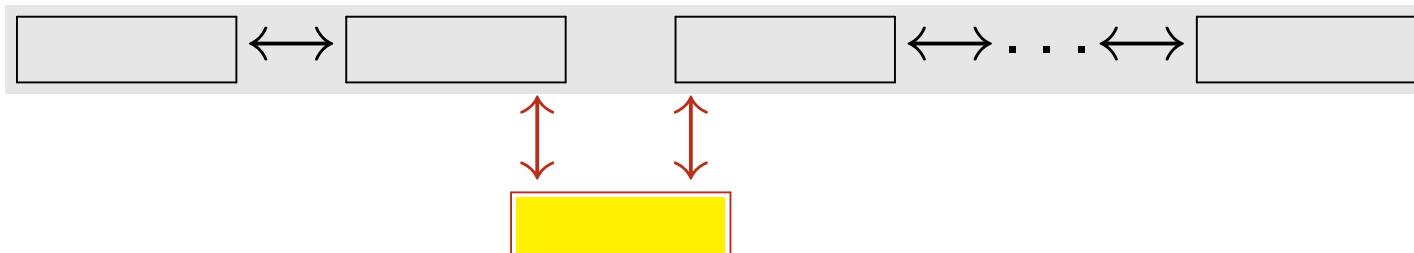


Klasy implementujące Collection (przykłady)



- **ArrayList** — jak tablice, ale automat. zmiana rozmiaru, wstawianie i usuwanie;
Uwaga: `ls[i]` nie działa, trzeba stosować `ls.get(i)` oraz `ls.set(i,x)` ;

- **LinkedList** — 
 - powolny dostęp do elementu zadanego indeksem (przeszukiwanie listy);
 - powiększanie lub zmniejszanie listy bez ograniczeń;
 - łatwe wstawianie w środek i usuwanie ze środka:



Zwykłe tablice, czy listy wiązane, czy listy tablicowe?

Jakie zbiory do jakiego problemu programistycznego?

Zwykłe tablice, czy listy wiązane, czy listy tablicowe?

Jakie zbiory do jakiego problemu programistycznego?

	zwykłe tablice	listy wiązane	listy tablicowe
stała liczba i ustalona kolejność elementów	✓		

Zwykłe tablice, czy listy wiązane, czy listy tablicowe?

Jakie zbiory do jakiego problemu programistycznego?

	zwykłe tablice	listy wiązane	listy tablicowe
stała liczba i ustalona kolejność elementów	✓		
pytania o element o dowolnym indeksie	✓		✓

Zwykłe tablice, czy listy wiązane, czy listy tablicowe?

Jakie zbiory do jakiego problemu programistycznego?

	zwykłe tablice	listy wiązane	listy tablicowe
stała liczba i ustalona kolejność elementów	✓		
pytania o element o dowolnym indeksie	✓		✓
zmienna liczba elementów		✓	✓

Zwykłe tablice, czy listy wiązane, czy listy tablicowe?

Jakie zbiory do jakiego problemu programistycznego?

	zwykłe tablice	listy wiązane	listy tablicowe
stała liczba i ustalona kolejność elementów	✓		
pytania o element o dowolnym indeksie	✓		✓
zmienna liczba elementów		✓	✓
wstawienia i usunięcia tylko z początku lub końca		✓	✓

Zwykłe tablice, czy listy wiązane, czy listy tablicowe?

Jakie zbiory do jakiego problemu programistycznego?

	zwykłe tablice	listy wiązane	listy tablicowe
stała liczba i ustalona kolejność elementów	✓		
pytania o element o dowolnym indeksie	✓		✓
zmienna liczba elementów		✓	✓
wstawienia i usunięcia tylko z początku lub końca		✓	✓
wstawienia i usunięcia w środku		✓	