

PROGRAMOWANIE OBIEKTOWE I GUI

dr inż. Michał Tomaszewski

katedra Metod Programowania
Polsko-Japońska Akademia Technik Komputerowych

Poruszyliśmy zagadnienia:

- miejsca definicji klas

Poruszyliśmy zagadnienia:

- miejsca definicji klas
- klasy abstrakcyjne

Plan wykładu:

- Interfejsy w wersji przed Java 8

Plan wykładu:

- Interfejsy w wersji przed Java 8
- Interfejsy po Java 8

Plan wykładu:

- Interfejsy w wersji przed Java 8
- Interfejsy po Java 8
- Interfejsy vs klasy abstrakcyjne

Plan wykładu:

- Interfejsy w wersji przed Java 8
- Interfejsy po Java 8
- Interfejsy vs klasy abstrakcyjne
- Wyrażenia Lambda

PRZYKŁAD 1

Klasa może dziedziczyć co najwyżej **jedną** klasę, ale może implementować dowolnie wiele interfejsów.

Klasa może dziedziczyć co najwyżej **jedną** klasę, ale może implementować dowolnie wiele interfejsów.

```
class Derived
    extends Primary
    implements Drawable, Movable {

    // ...
}
```

Interfejs może dziedziczyć dowolnie wiele interfejsów.

Interfejs może dziedziczyć dowolnie wiele interfejsów.

```
interface Drawable {  
    void draw();  
}  
  
interface Movable {  
    void moveTo(int x, int y);  
    void moveBack();  
}  
  
interface PrintableAndMovable  
    extends Drawable, Movable {  
  
    // ...  
}
```

INTERFEJSY CD

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach zmiennych.

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach zmiennych.

Zmienne zdefiniowane w interfejsie są domyślnie:

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach zmiennych.

Zmienne zdefiniowane w interfejsie są domyślnie:

- `public`

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach zmiennych.

Zmienne zdefiniowane w interfejsie są domyślnie:

- public
- static

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach zmiennych.

Zmienne zdefiniowane w interfejsie są domyślnie:

- public
- static
- final

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach zmiennych.

Zmienne zdefiniowane w interfejsie są domyślnie:

- `public` – aby był do nich dostęp z poziomu wszystkich klas;
- `static`
- `final`

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach zmiennych.

Zmienne zdefiniowane w interfejsie są domyślnie:

- `public` – aby był do nich dostęp z poziomu wszystkich klas;
- `static` – ponieważ `interface` nie fabrykuje obiektów, odwołanie do takiej zmiennej jest realizowane przez podanie `nazwyInterfejsu.nazwyZmiennej`
- `final`

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach zmiennych.

Zmienne zdefiniowane w interfejsie są domyślnie:

- `public` – aby był do nich dostęp z poziomu wszystkich klas;
- `static` – ponieważ `interface` nie fabrykuje obiektów, odwołanie do takiej zmiennej jest realizowane przez podanie `nazwyInterfejsu.nazwyZmiennej`
- `final` – aby uniknąć konfliktów gdy dwie klasy implementują ten sam interfejs

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach metod.

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach metod.

W takich metodach typ zwracany musi zostać poprzedzony specyfikatorem **default**.

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach metod.

W takich metodach typ zwracany musi zostać poprzedzony specyfikatorem **default**.

Zysk - ?

Począwszy od Java 8 istnieje możliwość deklarowania w interfejsach metod.

W takich metodach typ zwracany musi zostać poprzedzony specyfikatorem **default**.

Zysk - klasy pochodne nie muszą implementować ich ciała.

PRZYKŁAD 2

Co zatem gdy:

- interfejs **A** implementuje metodę `xxx()`,

Co zatem gdy:

- interfejs A implementuje metodę xxx(),
- interfejs B implementuje metodę xxx(),

Co zatem gdy:

- interfejs A implementuje metodę xxx(),
- interfejs B implementuje metodę xxx(),
- a klasa C implementuje oba interfejsy

Co zatem gdy:

- interfejs **A** implementuje metodę `xxx()`,
- interfejs **B** implementuje metodę `xxx()`,
- a klasa **C** implementuje oba interfejsy – ???

Co zatem gdy:

- interfejs **A** implementuje metodę `xxx()`,
- interfejs **B** implementuje metodę `xxx()`,
- a klasa **C** implementuje oba interfejsy – mamy problem!

PRZYKŁAD 3

W ciele interfejsu można również zadeklarować metody poprzedzone specyfikatorem `static`.

W ciele interfejsu można również zadeklarować metody poprzedzone specyfikatorem `static`.

Zatem, takie same fragmenty kodu wykonywane przez metody `default` mogą zostać wyizolowane do metod statycznych.

PRZYKŁAD 4

KLASY ABSTRAKCYJNE

Klasa abstrakcyjna to taka, która zawiera przynajmniej jedną metodę abstrakcyjną.

Klasa abstrakcyjna to taka, która zawiera przynajmniej jedną metodę nie zawierającą ciała.

Klasa abstrakcyjna to taka, w której słowo `class` poprzedza słowo kluczowe `abstract` i może zawierać abstrakcyjne metody nie zawierające ciał

- `interface` nie posiada nie statycznych pól

- `interface` nie posiada nie statycznych pól , podczas gdy klasa `abstract` może takie posiadać

- `interface` nie posiada nie statycznych pól , podczas gdy klasa `abstract` może takie posiadać
- `interface` nie może posiadać konstruktorów

- `interface` nie posiada nie statycznych pól , podczas gdy klasa `abstract` może takie posiadać
- `interface` nie może posiadać konstruktorów , a klasa `abstract` może

- `interface` nie posiada nie statycznych pól , podczas gdy klasa `abstract` może takie posiadać
- `interface` nie może posiadać konstruktorów , a klasa `abstract` może
- `interface` nie może posiadać metod `final`

- `interface` nie posiada nie statycznych pól , podczas gdy klasa `abstract` może takie posiadać
- `interface` nie może posiadać konstruktorów , a klasa `abstract` może
- `interface` nie może posiadać metod `final`, a klasa `abstract` może

WYRAŻENIA LAMBDA

Jeżeli interfejs zawiera dokładnie jeden nagłówek niezaimplementowanej metody, wówczas nazywamy go funkcyjnym.

Jeżeli interfejs zawiera dokładnie jeden nagłówek niezaimplementowanej metody, wówczas nazywamy go funkcyjnym.

I to wszystko?

Jeżeli interfejs zawiera dokładnie jeden nagłówek niezaimplementowanej metody, wówczas nazywamy go funkcyjnym.

tak

`(Argument List) ->{expression;}`

```
public  
    interface Drawable {  
        void draw();  
    };
```

```
public
    interface Drawable {
        void draw();
    };

// ....
Drawable drawable = new Drawable(){
    void draw (){
        // rysuje sie
    }
} ;
```

```
public
    interface Drawable {
        void draw();
    };

// ....
Drawable drawable = new Drawable(){
    void draw (){
        // rysuje sie
    }
} ;
```

```
public
    interface Drawable {
        void draw();
    };

// ....
Drawable drawable = new Drawable(){
    void draw (){
        // rysuje sie
    }
} ;
```

```
public
    interface Drawable {
        void draw();
    }

    Drawable drawable = (){
        // rysuje sie
    }
```

```
public
    interface Drawable {
        void draw();
    }

Drawable drawable = () -> {
    // rysuje sie
}
```



```
() -> {System.out.println("Hello");};
```

```
() -> {System.out.println("Hello");};  
(a) -> {System.out.println(a);}
```

```
() -> {System.out.println("Hello");};  
(a) -> {System.out.println(a);}  
(a, b) -> a+b;
```

PRZYKŁAD 5

DZIĘKUJĘ