PROGRAMOWANIE OBIEKTOWE I GUI

dr inż. Michał Tomaszewski

katedra Metod Programowania
Polsko-Japońska Akademia Technik Komputerowych

WYKŁAD 02

Poruszyliśmy zagadnienia:

- Interfejsy cd

WYKŁAD 02

Poruszyliśmy zagadnienia:

- Interfejsy cd
- Interfejsy po Java 8

WYKŁAD 02

Poruszyliśmy zagadnienia:

- Interfejsy cd
- Interfejsy po Java 8
- Interfejsy vs klasy abstrakcyjne

WYKŁAD 02

Poruszyliśmy zagadnienia:

- Interfejsy cd
- Interfejsy po Java 8
- Interfejsy vs klasy abstrakcyjne
- Wyrażenia Lambda

Plan wykładu:

- Typy generyczne

Plan wykładu:

- Typy generyczne
- Wykorzystanie interfejsów

Plan wykładu:

- Typy generyczne
- Wykorzystanie interfejsów
- Typy wylicznikowe

TYPY GENERYCZNE

Problem

```
public
    interface List<E>
    extends Collection<E>{

    // ...
};
```

TYPY GENERYCZNE - KONWENCJE

Kiedy korzystamy z typów generycznych?

TYPY GENERYCZNE - KONWENCJE

Typy generyczne głównie są wykorzystywane podczas definiowania bibliotek lub API.

TYPY GENERYCZNE - KONWENCJE

Przyjęte nazewnictwo:

- T - type

TYPY GENERYCZNE - KONWENCJE

Przyjęte nazewnictwo:

- T - type
- K - key

TYPY GENERYCZNE - KONWENCJE

Przyjęte nazewnictwo:

- T - type
- K - key
- N - number

TYPY GENERYCZNE - KONWENCJE

Przyjęte nazewnictwo:

- T - type
- K - key
- N - number
- E - element

TYPY GENERYCZNE - KONWENCJE

Przyjęte nazewnictwo:

- T - type
- K - key
- N - number
- E - element
- V - value

TYPY GENERYCZNE - KONWENCJE

Przyjęte nazewnictwo:

- T - type
- K - key
- N - number
- E - element
- V - value
- S, U, V, ... - drugi, trzeci i kolejne typy

TYPY GENERYCZNE - KONWENCJE

Przyjęte nazewnictwo:

- T - type
- K - key
- N - number
- E - element
- V - value
- S, U, V, ... - drugi, trzeci i kolejne typy

```
public
    interface Pair<K, V> {
    public K getKey();
    public V getValue();
};
```

Przykład

WYKORZYSTANIE INTERFEJSÓW

PRZYKŁAD PRAKTYCZNY

- przyjmijmy że chcemy opisać *Studenta*;

PRZYKŁAD PRAKTYCZNY

- przyjmijmy że chcemy opisać Studenta;
- przyjmijmy że chcemy poukładać *Student*ów w zadanej kolejności

PRZYKŁAD PRAKTYCZNY

- przyjmijmy że chcemy opisać Studenta;
- przyjmijmy że chcemy poukładać Studentów w zadanej kolejności
- only one answer - ?

PRZYKŁAD PRAKTYCZNY

- przyjmijmy że chcemy opisać Studenta;
- przyjmijmy że chcemy poukładać Studentów w zadanej kolejności
- only one answer - `Arrays.sort`

COMPARABLE VS COMPARATOR

- interfejs `Comparable` jest wykorzystywany gdy chcemy posortować elementy tylko w jeden sposób, podczas gdy `Comparator` pozwala na określenie sposobu sortowania

COMPARABLE VS COMPARATOR

- interfejs `Comparable` jest wykorzystywany gdy chcemy posortować elementy tylko w jeden sposób, podczas gdy `Comparator` pozwala na określenie sposobu sortowania
- aby użyć interfejsu `Comparable` musimy go zdefiniować podczas tworzenia klasy, co nie jest konieczne przy wykorzystaniu `Comparator`-a

COMPARABLE VS COMPARATOR

- interfejs `Comparable` jest wykorzystywany gdy chcemy posortować elementy tylko w jeden sposób, podczas gdy `Comparator` pozwala na określenie sposobu sortowania
- aby użyć interfejsu `Comparable` musimy go zdefiniować podczas tworzenia klasy, co nie jest konieczne przy wykorzystaniu `Comparator`-a
- `Comparable` zdefiniowano w pakiecie `java.lang`, a `Comparator` w pakiecie `java.util`

Przykład

TYPY WYLICZNIKOWE

TYPY WYLICZNIKOWE - CZYM SĄ?

Typ wylicznikowy zapisem skończonej ilości wartości, z których każda ma swoją własną (najczęściej znaczącą) nazwę.

TYPY WYLICZNIKOWE - SKŁADNIA

```
[public] enum EnumerationTypeName {
    constant1, constant2, ...;

}
```

Przykład

TYPY WYLICZNIKOWE

Po co:

TYPY WYLICZNIKOWE

Po co:

- dla predefiniowanych list wartości,

TYPY WYLICZNIKOWE

Po co:

- dla predefiniowanych list wartości,
- gdy zmiennej może być nadana wartość z ograniczonego zestawu,

TYPY WYLICZNIKOWE

Po co:

- dla predefiniowanych list wartości,
- gdy zmiennej może być nadana wartość z ograniczonego zestawu,
- używając enum-ów zwiększamy czas kompilacji, ale unikamy błędów przekazywanych wartości.

TYPY WYLICZNIKOWE - OGRANICZENIA

- Wszystkie typy wyliczenikowe dziedziczą po `java.lang.Enum`

TYPY WYLICZNIKOWE - OGRANICZENIA

- Wszystkie typy wyliczniknikowe dziedziczą po `java.lang.Enum`
- `Enum` wykorzystuje własną przestrzeń nazewniczą

TYPY WYLICZNIKOWE - OGRANICZENIA

- Wszystkie typy wyliczenikowe dziedziczą po `java.lang.Enum`
- `Enum` wykorzystuje własną przestrzeń nazewniczą
- wartości `Enum` są z definicji `statyczne` i `final`

TYPY WYLICZNIKOWE - OGRANICZENIA

- Wszystkie typy wylicznikowe dziedziczą po `java.lang.Enum`
- `Enum` wykorzystuje własną przestrzeń nazewniczą
- wartości `Enum` są z definicji `statyczne` i `final`
- wartości `Enum` mogą być bezpiecznie porównywane:

TYPY WYLICZNIKOWE - OGRANICZENIA

- Wszystkie typy wylicznikowe dziedziczą po `java.lang.Enum`
- `Enum` wykorzystuje własną przestrzeń nazewniczą
- wartości `Enum` są z definicji `statyczne` i `final`
- wartości `Enum` mogą być bezpiecznie porównywane:
  - w wyrażeniach `switch - case`

TYPY WYLICZNIKOWE - OGRANICZENIA

- Wszystkie typy wyliczniknowe dziedziczą po `java.lang.Enum`
- `Enum` wykorzystuje własną przestrzeń nazewniczą
- wartości `Enum` są z definicji `statyczne` i `final`
- wartości `Enum` mogą być bezpiecznie porównywane:
  - w wyrażeniach `switch - case`
  - przez operator `==`

TYPY WYLICZNIKOWE - OGRANICZENIA

- Wszystkie typy wyliczniknowe dziedziczą po `java.lang.Enum`
- `Enum` wykorzystuje własną przestrzeń nazewniczą
- wartości `Enum` są z definicji `statyczne` i `final`
- wartości `Enum` mogą być bezpiecznie porównywane:
  - w wyrażeniach `switch - case`
  - przez operator `==`
  - metodą `equals()`

Dziękuję