Programowanie obiektowe

Wykład 7:
Typy uogólnione. Stringi.

Jak operować na danych różnych typów?

- Wymagana zgodność typów
- Rzutowanie w górę
  - Tylko na typ bazowy
  - Bezpieczne
- Rzutowanie w dół:
  - Tylko na typ „oryginalny” lub bazowy typu oryginalnego
  - Może spowodować błędy!!!
- A może typ jako parametr??

Przykład: uniwersalny „pojemnik” na obiekty

```
public class Box {
  private Object object;
  public void add(Object
       object) {
    this.object = object;
  }
  public Object get() {
    return object;
  }
}
```

```
Box integerBox = new Box();
integerBox.add(new Integer(10));
Integer someInteger = (Integer) integerBox.get();
System.out.println(someInteger);
```

```
Box integerBox = new Box();
integerBox.add("10");
Integer someInteger=(Integer)integerBox.get();
System.out.println(someInteger);
```

```
Exception in thread "main"
  java.lang.ClassCastException:
    java.lang.String cannot be cast to java.lang.Integer
      at BoxDemo.main(BoxDemo.java:6)
```


Klasy sparametryzowane typami

- class Nazwa<T1, T2, T3> { …
- Nazwa<Integer, String, Float> o;
- new Nazwa<Integer, String, Float>();
- Typ musi być klasą

Przykład: uogólniony „pojemnik” na obiekty

```
public class Box <T>{
  private T object;
 public void add(T
       object) {
   this.object = object;
  }
  public T get() {
    return object;
  }
}
```

```
Box<Integer> integerBox = new Box<Integer>();
integerBox.add(new Integer(10));
Integer someInteger = integerBox.get();
System.out.println(someInteger);
```

```
Box<Integer> integerBox = new Box<Integer>();
integerBox.add("10");  // Błąd kompilacji
Integer someInteger=integerBox.get();
System.out.println(someInteger);
```


Dziedziczenie po klasach sparametryzowanych typami

public class Para<T1, T2>

- Dziedziczenie po klasie ukonkretnionej:
  public class ParaInt extends Para<Integer, Integer> {…}
- Dziedziczenie po klasie sparametryzowanej:
  public class ParaJednorodna<T3> extends Para<T1,T2>{…}

Metody parametryzowane typami

- public <U> void metoda(U u){ …}
- o.<Integer>metoda(5);

Ograniczenia na typy sparametryzowane

- <U extends Number>

  U musi być klasą dziedziczącą po Number
- <U implements MojInterfejs>
- <U extends Number & MojInterfejs>

Typ nieznany: ?

- <? extends T>
- <?> == <? extends Object>
- <? super T>

Podsumowanie typów uogólnionych

- Typy proste nie mogą być parametrami
  - Ale jest autoboxing (od Java 1.5):

```
int i1;
Integer i2=new Integer();
i1=i2;    // dawniej:  i1=i2.value;
I2=i1;   // dawniej : i2.value=i1;
```

- Nie można tworzyć obiektów typu będącego parametrem:

```
class Klasa<T>{  T o=new T(); …
```

- Rzutowanie na typ uogólniony generuje ostrzeżenia:

```
pi = (Para<Integer, Integer>) p;
```

- Typy uogólnione są konwertowane podczas kompilacji na typy „surowe”

Przykład 1

```
class FArray {
  public static <T> T[ ] fill(T[ ] a, Generator<T> gen) {
    for(int i = 0; i < a.length; i++)   a[i] = gen.next();
    return a;
  }
}

public class PrimitiveGenericTest {
  public static void main(String[ ] args) {
    String[ ] strings = FArray.fill(
      new String[7], new RandomGenerator.String(10));
    for(int i=0; i<strings.length; i++)
      System.out.println(strings[i]);
    Integer[ ] integers = FArray.fill(
      new Integer[7], new RandomGenerator.Integer());
    for(int i=0; i<integers.length; i++)
      System.out.println(integers[i]);
      // int[ ] b = FArray.fill(new int[7], new RandIntGenerator());
  }
}
```


Przykład 2

```
public class Erased<T> {
  private final int SIZE = 100;
  public static void f(Object arg) {
    T var = new T();
    T[ ] array = new T[SIZE];
    T[ ] array = (T)new Object[SIZE]; // Unchecked warning
  }
}
```

Przykład 3

```
class GenericBase<T> {
  private T element;
  public void set(T arg) { arg = element; }
  public T get() { return element; }
}

class Derived1<T> extends GenericBase<T> {}

class Derived2 extends GenericBase {}

// class Derived3 extends GenericBase<?> {}
// Strange error:
//   unexpected type found : ?
//   required: class or interface without bounds


public class ErasureAndInheritance {
  @SuppressWarnings("unchecked")
  public static void main(String[] args) {
    Derived2 d2 = new Derived2();
    Object obj = d2.get();
    d2.set(obj);      // Warning
  }
}
```


String

- Nie można zmienić długości (np. dodać znaki, usunąć znaki itp.)
- Metody: charAt(), concat(), indexOf(), length(), replace(), toUpperCase(), toLowerCase(), substring() itp.
- Literały też są obiektami:
  - np. ”Napis”.length()
- Operator „+” jako konkatenacja:
  - ”string” + 5 -> „string5”
  - 5+”string” ->?

StringBuilder

- Modyfikowalny string
- Metody: append(), delete(), insert(), reverse(), itp.
- StringBuffer - wersja dla programów wielowątkowych.

Pytania

1. Jakie konstrukcje (klasy, interfejsy, parametry metod, wartość zwracana przez metodę, pole klasy, zmienna lokalna, obiekt) mogą być sparametryzowane typem?
2. Jakie są ograniczenia typów uogólnionych?
3. Jaka jest różnica pomiędzy typami uogólnionymi a rzutowaniem?

Koniec