# Шаблони(Generics) и приложението им при работа с базовите колекции.

Николай Вълчев, Петър Казаков / SAP Labs Bulgaria
Март 2013/Ноември 2014

Public

# Шаблони – дефиниция и история

- Дефиниция: Стил в програмирането при който алгоритмите се имплементират с типове/класове, които се инстанциират по-късно като параметри;
- Въведени в ML/1973, по-късно в CLU, Scheme и Ada;
- Дадени: N структури данни, M алгоритъма. Цел: Да имплементираме N+M комбинации от алгоритми и структури данни;
- Шаблоните са теоретично свързани с Банаховите пространства.

# Шаблони - въведение

- Добавени в Java 5.
- Придават устойчивост към програмите, като осигуряват проверка по време на компилация за съвместимост между типовете.
- Дават възможност за параметризиране чрез типове на класове и методи.
  - Възможност за преизползване на код с различни цели.

# Шаблонни типове

- Клас или интерфейс, който е параметризиран чрез параметър за тип.
- Параметърът за тип може да приема кой да е непримитивен тип – клас, интерфейс, масив и т.н.
- Дефиниция
  - class name<T1, T2, ..., Tn> { /* ... */ }

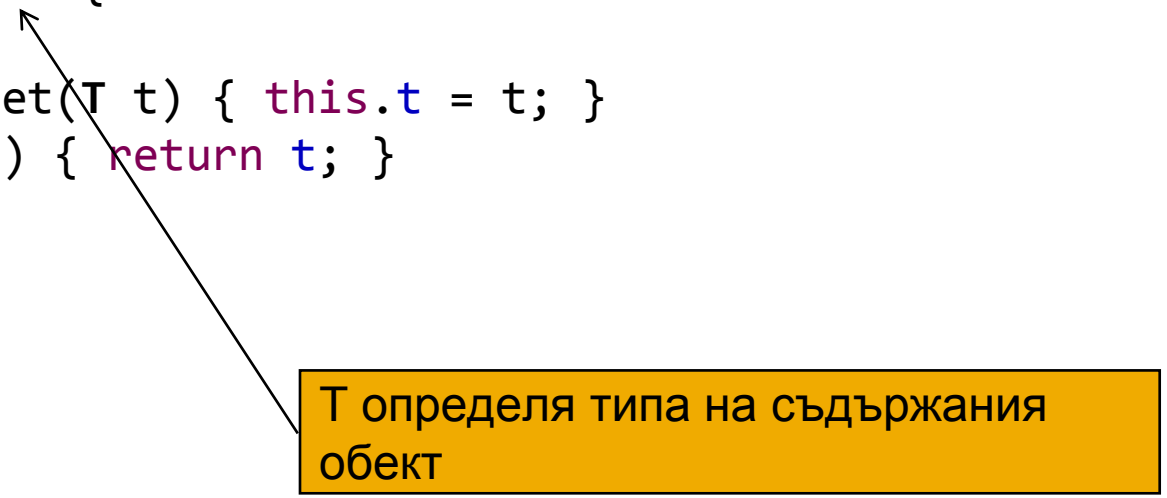
# Не-шаблонна кутия ☺

- Съдържа какъв да е обект

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

# И сега - шаблонна кутия ☺

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



# Шаблонни типове

- Конвенция за именуване на параметрите за тип:
  - E - Element
  - K - Key
  - N - Number
  - T - Type
  - V - Value
  - S,U,V etc. - 2nd, 3rd, 4th types

# Малко встрани – Autoboxing & Unboxing

- Програмният език Java дефинира обгръщащи класове за всички примитивни типове.
- Autoboxing е автоматичното конвертиране, което Java компилатора прави между примитивните типове и техните обгръщащи типове.

```
List<Integer> list = new ArrayList<>();
for (int i = 1; i < 50; i += 2)  { list.add(i); }
```

- Обратната конверсия – от обгръщащ тип към примитивен се нарича unboxing.

```
public static int sumEven(List<Integer> li) {
        int sum = 0;
        for (Integer i: li) if (i % 2 == 0) { sum += i; }
        return sum;
    }
```

| Примитивен тип | Обгръщащ тип |
|---|---|
| boolean | Boolean |
| byte | Byte |
| char | Character |
| float | Float |
| int | Integer |
| double | Double |
| long | Long |
| short | Short |

# Шаблонни класове. Създаване на инстанции

- Дългият начин:
  - Box<Integer> integerBox = new Box<Integer>();
- Diamond оператор <> - след Java 1.7
  - Box<Integer> integerBox = new Box<>();

# Сурови типове (Raw types)

- Името на шаблонен клас или интерфейс без аргументите за тип.
- Избягвайте ги:
  - Отнемат възможността на компилатора да открива грешки по време на компилация.
- Имаме:

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

- Raw type на Box<T> е Box.
- Можем да създадем инстанция по следния начин
  - Box rawBox = new Box();

# Сурови типове (Raw types)

- Преди Java 5 практически всички типове са били „сурови“.
- Съвместимостта със стари версии е естествена:
  - Може безопасно да се присвои инстанция на параметризиран тип на суровия му тип:

```
Box<String> stringBox = new Box<>();
Box rawBox = stringBox;
```

- Обратното присвояване се компилира с предупреждение:

```
Box rawBox = new Box(); // rawBox is a raw type of Box<T>
Box<Integer> intBox = rawBox; // warning: unchecked conversion
```

- Също се генерира предупреждение ако се опитаме да изпълним шаблонен метод пред инстанция на суров тип:

```
Box<String> stringBox = new Box<>();
Box rawBox = stringBox;
rawBox.set(8); // warning: unchecked invocation to set(T)
```

# Сурови типове (Raw types) 2

Vector<String> strings = **new Vector<String>(10);**

Vector<Object> objects = (Vector<Object>)strings; //Compile-time error

Vector<String> strings = **new Vector<String>(10);**

Vector objects = strings;

objects.add(**new Object());**

Object anObject = strings.get(0);

String aString = strings.get(0);// Class cast exception

# Шаблонни методи

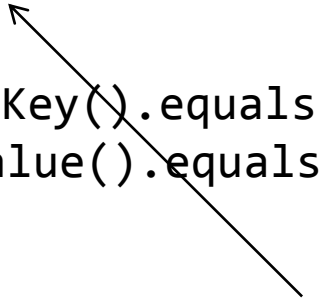
- Могат да използват нови параметри за тип, недекларирани от класа.
- Новите параметри за тип за видими единствено за метода, който ги декларира
- Могат да са
  - статични;
  - на инстанцията;
  - конструктури.

# Шаблонни методи – примери (двойка)

```
public class Pair<K, V> {
    private K key;
    private V value;
    // Generic constructor
    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }
    // Generic methods
    public void setKey(K key) { this.key = key; }
    public void setValue(V value) { this.value = value; }
    public K getKey()   { return key; }
    public V getValue() { return value; }
}
```

# Шаблонни методи – примери

```
public class Util {
    // Generic static method
    public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V>
p2) {
        return p1.getKey().equals(p2.getKey()) &&
            p1.getValue().equals(p2.getValue());
    }
}
```



Параметрите за тип трябва да са преди типа на връщаната стойност

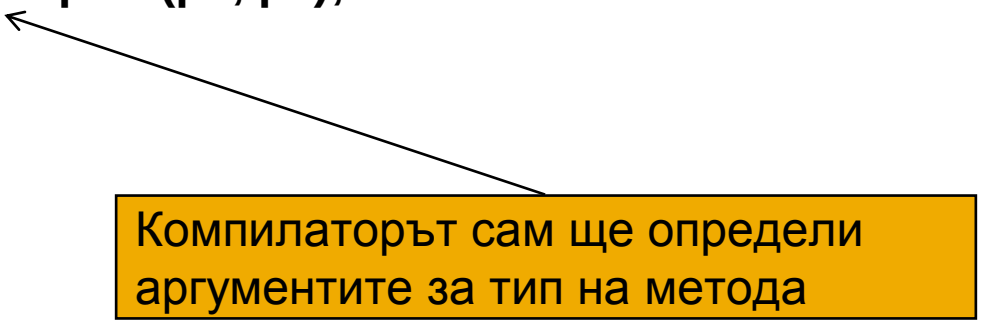
# Шаблонни методи - извикване

- Пълен синтаксис:

```
Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");
boolean same = Util.<Integer, String>compare(p1, p2);
```

- Кратък синтаксис:

```
Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");
boolean same = Util.compare(p1, p2);
```



# Ограничени параметри за тип

- Използват се за ограничаване на възможните стойности на параметрите за тип.
- Синтаксис:

  Class A { /* ... */ }

  interface B { /* ... */ }

  interface C { /* ... */ }

  class D **<T extends A & B & C>** { /* ... */ }
- Правят възможна имплементацията на алгоритми с широко приложение.

# Алгоритми с широко предназначение

**Пример:**

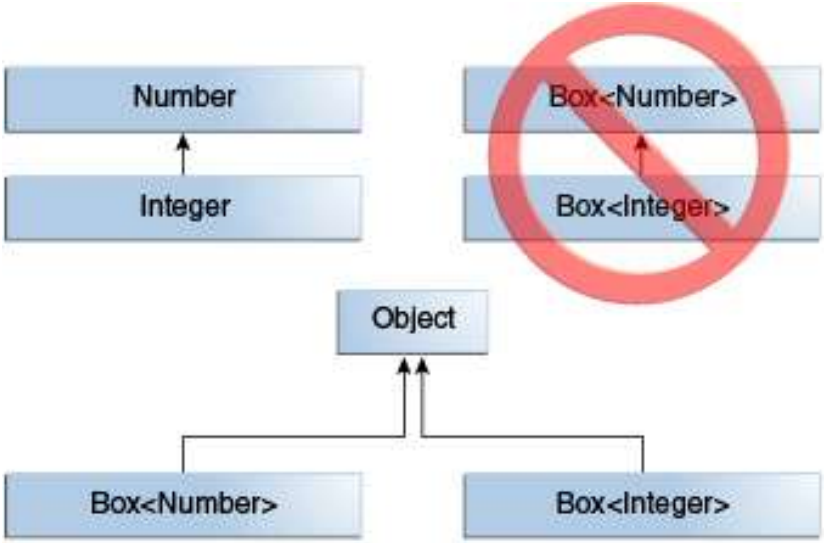
```
public static <T extends Comparable<T>> int countGreaterThan(T[]
anArray, T elem) {
        int count = 0;
        for (T e : anArray)
            if (e.compareTo(elem) > 0)
                ++count;
        return count;
    }
```

# Наследяване

- Integer is-a Object
- Integer is-a Number
- Double is-a Number

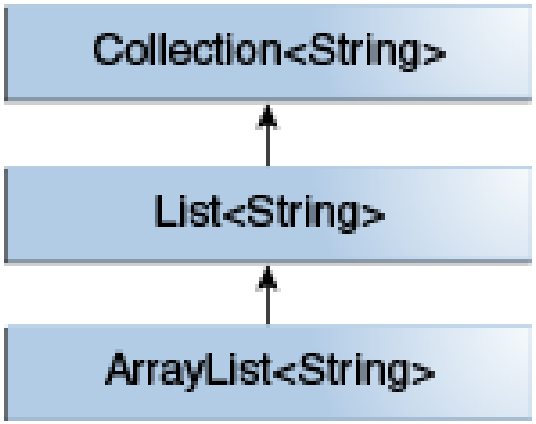
**BUT**

- Box<Integer> is-not-a Box<Number>, техният общ производен клас е Object



# Шаблони и подтипове

- Подтип на шаблонен клас или интерфейс получаваме чрез разширяване или имплементиране.
- Ако **аргумента за тип е еднакъв**, то is-a връзката е в сила.
- Пример от Collections framework:



# Wildcards

- ? – означава неизвестен тип
- Може да се използва за тип на:
  - параметър;
  - поле на инстанцията;
  - локална променлива;
  - тип на връщане.
- Неможе да се използва за аргумент за тип при извикване на:
  - шаблонен метод;
  - създаването на инстанция на шаблонен клас.

# Wildcards, ограничени отгоре

- Ако искате да създадете метод, който намира сумата на елементите в списък от Integer, Double, Float, Number.
- Гъвкавото решение:

```
public static double sumOfList(List<? extends Number> list) {
        double sum = 0.0;
        for (Number n : list)
            sum += n.doubleValue();
        return sum;
}
```

Използваме метод дефиниран в java.lang.Number

- Препоръчва се при дефинирането на входни параметри за метода.
- Read-only семантика.

# Неограничени wildcards

- Списък от елементи от неизвестен тип - List<?>
- List<?> е еквивалентно на List<? extends Object>
- Използва се когато:
  - Функционалността която пишем може да се имплементира единствено със знанието за методите в java.lang.Object.
  - Не се интересуваме от типа на елементите в списъка, а се интересуваме от характеристики на самия списък – например размер на списъка.

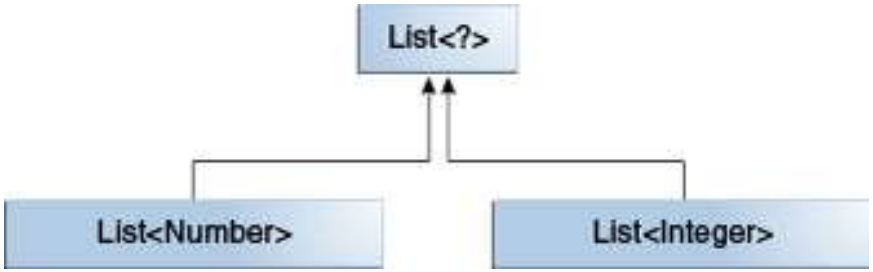
# Wildcars, ограничени отдолу

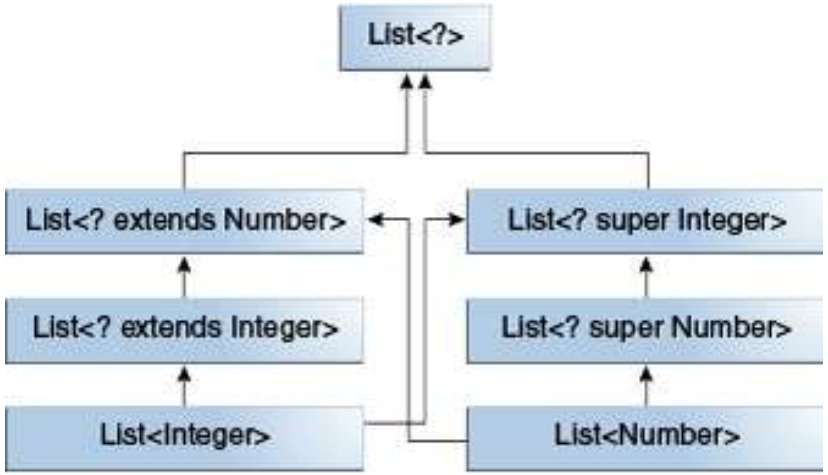
- Искаме да създадем метод, които да добавя Integer обекти към списък.
- Искаме метода да работи с колекции от Integer, Number, Object.
- Решението:

```
public static void addNumbers(List<? super Integer> list) {
        for (int i = 1; i <= 10; i++) {
            list.add(i);
        }
}
```

- Препоръчва се при деклариране на изходни параметри.

# Wildcards и подтипове



# Изтриване на информацията за типовите аргументите по време на изпълнение

- Java компилаторът изтрива информацията за типовите аргументи и тази информация не е налична по време на изпълнение:
  - Всички типови параметри в шаблонни класове и интерфейси се заместват:
    - Ако са неограничени – с Object:

Пример:

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

След заместване става:

```
public class Box {
    private Object t;
    public void set(Object t) { this.t = t; }
    public Obeject get() { return t; }
}
```

# Изтриване на информацията за типовите аргументите по време на изпълнение

- Ако са ограничени – с техния ограничителен тип.

Нека имаме следната класова йерархия:

**class Shape { /* ... */ }**

**class Circle extends Shape { /* ... */ }**

**class Rectangle extends Shape { /* ... */ }**

Нека имаме дефиниран следния метод, който рисува дадена фигура:

**public static <T extends Shape> void draw(T shape) { /* ... */ }**

След заместването на типовият параметър от компилатора се получава:

**public static void draw(Shape shape) { /* ... */ }**

# Изтриване на информацията за типовите аргументите по време на изпълнение

- Компилаторът може да добави type casts ако е необходимо за запазването на консистентността на типовете.
- Може да се генерират **синтетични методи** с цел запазване на полиморфичните свойства при наследени шаблонни класове и имплементирани шаблонни интерфейси.
  - **Пример …..**

# Ограничения при шаблонните класове и интерфейси

- Немогат да се създават инстанции от типов параметър:

```
public static <E> void append(List<E> list) {
    E elem = new E(); // compile-time error
    list.add(elem);
}
```

- Немогат да се декларират статични полета на клас от типа на типов параметър.

```
public class MobileDevice<T> {
    private static T os; // compile-time error
    // ...
}
```

# Ограничения при шаблонните класове и интерфейси

- Немогат да се правят конвертирания м/у типове(casts), както и instanceof оператора с шаблонни типове.

```
public static <E> void rtti(List<E> list) {
        if (list instanceof ArrayList<Integer>) {  // compile-time error
            // ...
        }
}
```

- Немогат да се декларират масиви от параметризиран тип.

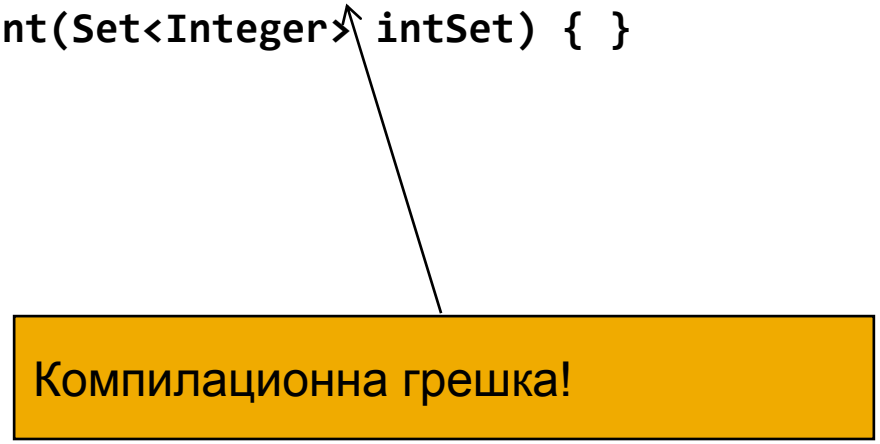
```
List<Integer>[] arrayOfLists = new List<Integer>[2];  // compile-time error
```

```
Object[] stringLists = new List<String>[];  // compiler error, but pretend it's allowed
stringLists[0] = new ArrayList<String>();   // OK
stringLists[1] = new ArrayList<Integer>();  // An ArrayStoreException should be thrown,
                                              // but the runtime can't detect it.
```

# Ограничения при шаблонните класове и интерфейси

- Неможе да се дефинират два метода с формални параметри, които след изтриване на типовите параметри имат еднакви сигратури

```
public class Example {
        public void print(Set<String> strSet) { }
        public void print(Set<Integer> intSet) { }
}
```



# Изключения. Какво са изключенията?

- Отбелязват събитие, което възникна по време на изпълнение и не е свързано с нормалната работа на приложението.
- Всеки метод, който съдържа логика, която може да доведе до изключение трябва:
  - да обработи изключението или
  - да декларира че може да „хвърли“ изключение от съответния тип

# Благодаря за вниманието!

nikolay.valchev@sap.com
peter.kazakov@sap.com

# ИЗПОЛЗВАНА ЛИТЕРАТУРА

- Java Tutorial: Generics
- Java Tutorial: Exceptions