КОЛЕКЦИИ И ШАБЛОННИ ТИПОВЕ

Структури от данни в Java

30.10.2018

ПРЕДНАТА ЛЕКЦИЯ ГОВОРИХМЕ ЗА:

- Wrapper types
- Статични член променливи и статични методи
- Enums
- Изключения

ДНЕС ПРОДЪЛЖАВАМЕ С:

- Колекции (collections)
- Шаблонни типове (generics)

КОЛЕКЦИИ

КОЛЕКЦИИ

- Java предоставя т.нар. collections framework, съдържащ интерфейси, имплементации и алгоритми върху най-използваните структури от данни.
- Всички* интерфейси и класове се намират в пакета java.util.

Някои ползи от наличието на collections framework:

- Не се налага да преоткриваме топлата вода
- Увеличават се скоростта и качеството на програмите ни
- Стимулира се преизползването на код

ИНТЕРФЕЙСИ ITERATOR И ITERABLE

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove();
}

public interface Iterable<T> {
    Iterator<T> iterator();
}
```

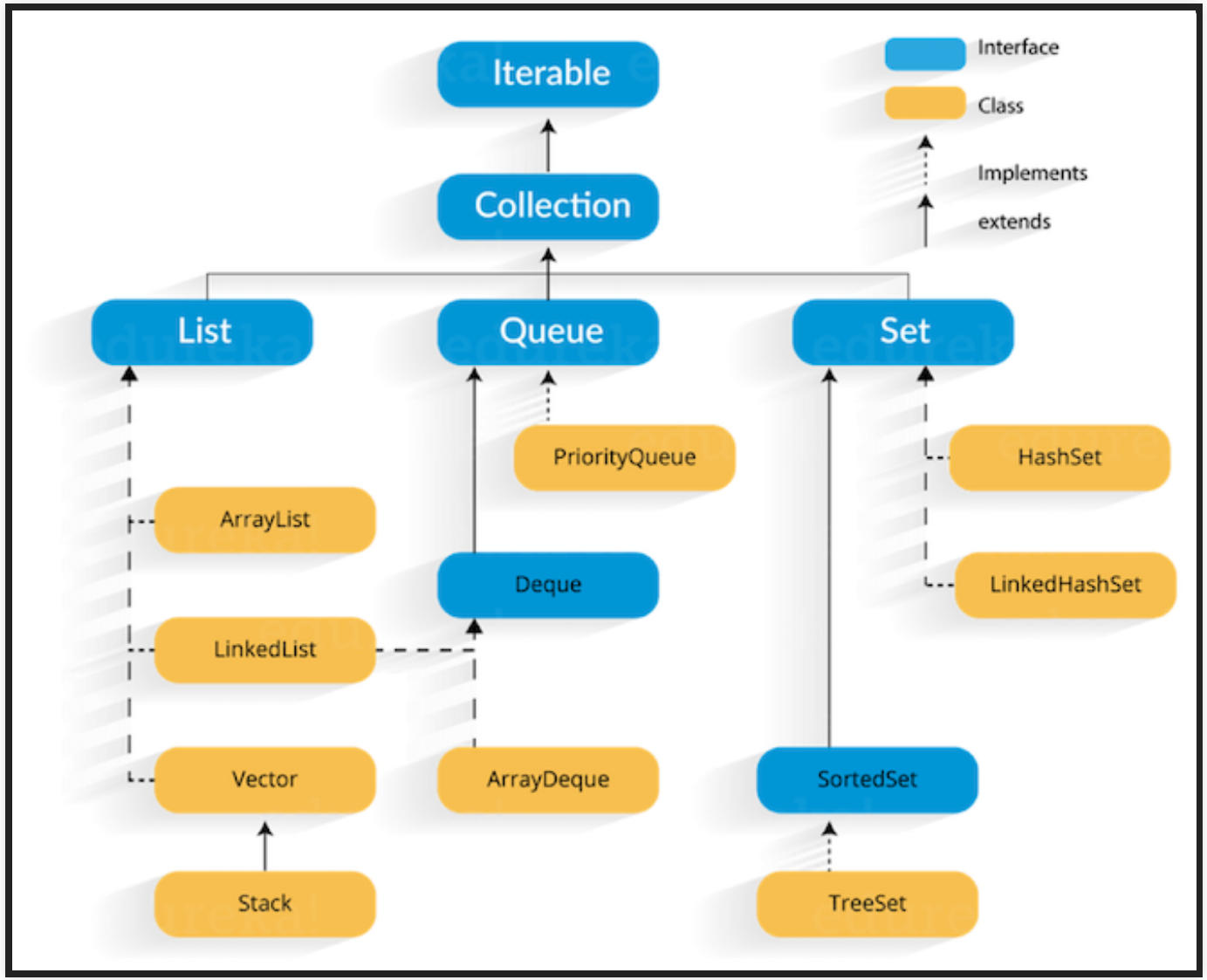
ИНТЕРФЕЙСИ ITERATOR И ITERABLE

- Итераторите предоставят унифициран начин за обхождане на елементите на дадена колекция.
- Колекциите (както и масивите) могат да се обхождат с foreach loop
- ConcurrentModificationException в еднонишков код?!

ОСНОВНИ СТРУКТУРИ ОТ ДАННИ

Основните структури от данни, използвани в имплементациите на колекциите са

- Масиви
- Свързани списъци
- Хеш таблици
- Дървета


Iterable
Collection
List
Queue
Set
Interface
Class
Implements
extends
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

ОБХОЖДАНЕ НА КОЛЕКЦИИ

```
List<Float> nums = Arrays.asList(4.999f, 0.271f, 7.1f, -1f);
```

- Чрез enhanced for-loop:

```
for (float current : nums) {
    System.out.printf("%.2f%n", current);
}
```

- Чрез итератор:

```
Iterator<Float> iterator = nums.iterator();
while (iterator.hasNext()) {
    System.out.printf("%.2f%n", iterator.next());
}
```

ОБХОДЖДАНЕ НА MAP

Според това какво ни е нужно, може да вземем от Map-а:

- множеството от ключовете

```
Set<Integer> keys = map.keySet();
```

- колекция от стойностите

```
Collection<String> values = map.values();
```

- колекция от двойките ключ-стойност

```
Set<Entry<Integer, String>> s = map.entrySet(
```

java.util.Collection

```
int size()
boolean isEmpty()
boolean contains(Object element)
boolean add(E element)
boolean remove(Object element)
Iterator<E> iterator()

boolean containsAll(Collection<?> c)
boolean addAll(Collection<? extends E> c)
boolean removeAll(Collection<?> c)
boolean retainAll(Collection<?> c)
void clear()

Object[] toArray()
<T> T[] toArray(T[] a)
```

List

```
boolean add(E e)
boolean contains(Object o)
E get(int index)
int indexOf(Object o)
boolean remove(Object o)
E remove(int index)
int size()
boolean isEmpty()
Object[] toArray()
List<E> subList(int fromIndex, int toIndex)
```

ИМПЛЕМЕНТАЦИИ НА LIST

- ArrayList - resize-ващ се масив
- LinkedList - двойно свързан списък
- Vector - resize-ващ се масив. Synchronized. Legacy
- Stack - наследява Vector. Legacy

АЛГОРИТМИЧНА СЛОЖНОСТ НА ОСНОВНИТЕ ОПЕРАЦИИ

	get	add	contains	next	remove(0)	iterator.remove
ArrayList	$O(1)$	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$
LinkedList	$O(n)$	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$

Queue

```
boolean add(E e)

// Retrieves, but does not remove, the head of the queue
E peek()

// Retrieves and removes the head of the queue
// Returns null if the queue is empty
E poll()

// Retrieves and removes the head of the queue
// Throws NoSuchElementException if the queue is empty
E remove()
```

ИМПЛЕМЕНТАЦИИ НА QUEUE

- PriorityQueue - heap (пирамида)
- LinkedList
- ArrayDeque - resize-ващ се масив

АЛГОРИТМИЧНА СЛОЖНОСТ НА ОСНОВНИТЕ ОПЕРАЦИИ

	offer	peek	poll	size
`PriorityQueue`	$O(\log n)$	$O(1)$	$O(\log n)$	$O(1)$
`LinkedList`	$O(1)$	$O(1)$	$O(1)$	$O(1)$
`ArrayDeque`	$O(1)$	$O(1)$	$O(1)$	$O(1)$

Set

```
boolean add(E e)
boolean contains(Object o)
boolean remove(Object o)
int size()
boolean isEmpty()
Object[] toArray()
```

ИМПЛЕМЕНТАЦИИ НА SET

- TreeSet - TreeMap. Червено-черно дърво
- HashSet - хеш таблица
- LinkedHashSet - хеш таблица + свързан списък
- EnumSet - битов масив

КОНСТРУКТОРИ НА HASHSET

```
HashSet(); // default initial capacity (16) and load factor (0
HashSet(Collection<? extends E> c);
HashSet(int initialCapacity);
HashSet(int initialCapacity, float loadFactor);
```

КОНСТРУКТОРИ НА TREESET

```
TreeSet(); // natural ordering
TreeSet(Collection<? extends E> c);
TreeSet(Comparator<? super E> comparator);
TreeSet(SortedSet<E> s);
```

java.lang.Comparable vs java.util.Comparator

```
public interface Comparable<T> {
    public int compareTo(T o);
}

public interface Comparator<T> {
    public int compare(T o1, T o2);
}
```

LINKEDHASHSET

АЛГОРИТМИЧНА СЛОЖНОСТ НА ОСНОВНИТЕ ОПЕРАЦИИ

	add	contains	next
HashSet	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashSet	$O(1)$	$O(1)$	$O(1)$
EnumSet	$O(1)$	$O(1)$	$O(1)$
TreeSet	$O(\log n)$	$O(\log n)$	$O(\log n)$

Операции над множества с **Set**

```
Set<String> one = new HashSet<>();
one.add("foo");
one.add("bar");
Set<String> two = new HashSet<>();
two.add("foo");
two.add("baba");

Set<String> union = new HashSet<>(one);
union.addAll(two); // union = [baba, bar, foo]

Set<String> intersection = new HashSet<>(one);
intersection.retainAll(two); // intersection = [foo]

Set<String> difference = new HashSet<>(one);
difference.removeAll(two); // difference = [bar]
```

Map

```
V put(K key, V value)
V get(Object key)
V remove(Object key)
boolean containsKey(Object key)
int size()
boolean isEmpty()
Set<K> keySet()
Collection<V> values()
```

ИМПЛЕМЕНТАЦИИ НА MAP

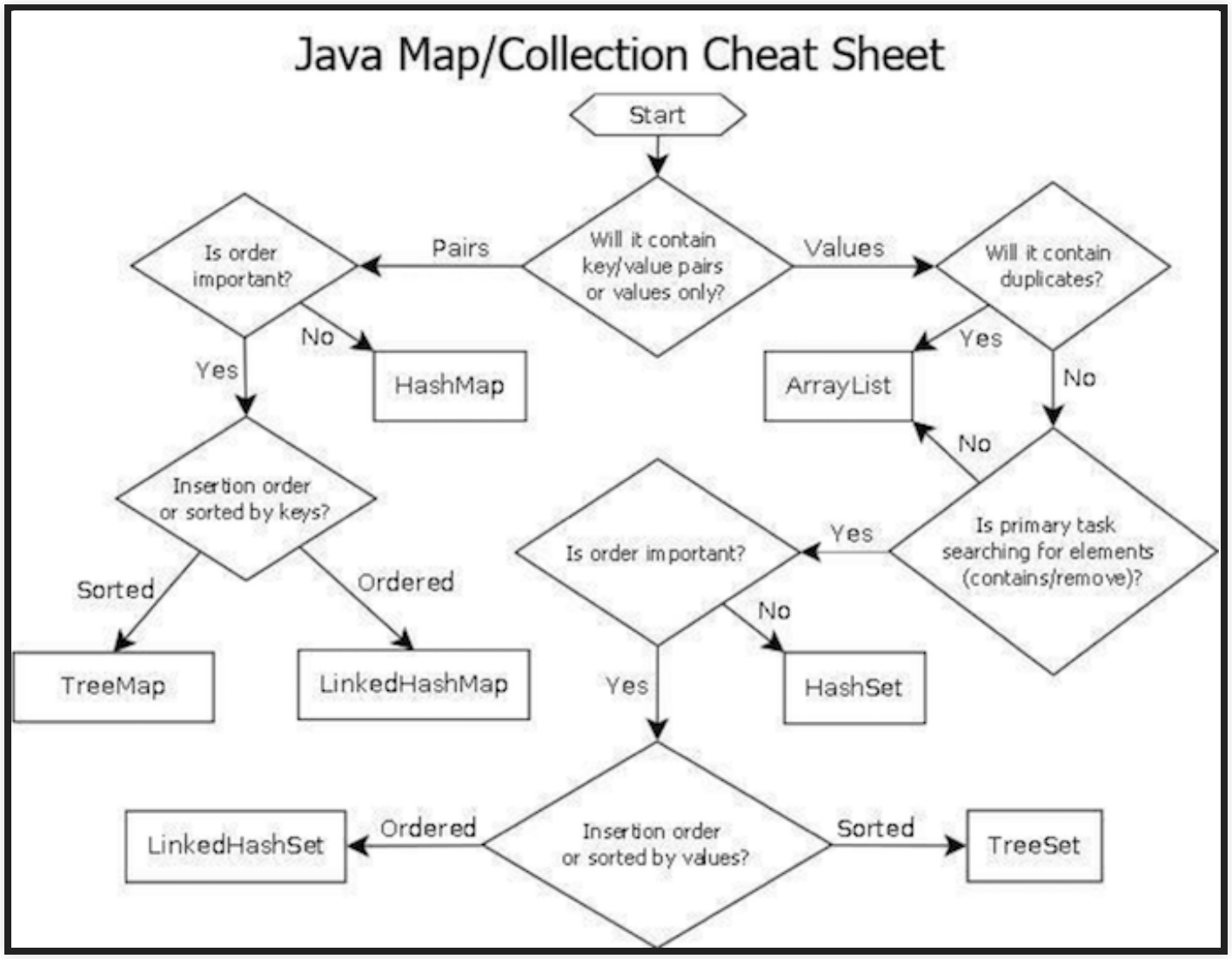
- HashMap - хеш таблица
- LinkedHashMap - хеш таблица + свързан списък
- EnumMap - битов масив
- TreeMap - червено-черно дърво

АЛГОРИТМИЧНА СЛОЖНОСТ НА ОСНОВНИТЕ ОПЕРАЦИИ

	get	containsKey	next
HashMap	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashMap	$O(1)$	$O(1)$	$O(1)$
EnumMap	$O(1)$	$O(1)$	$O(1)$
TreeMap	$O(\log n)$	$O(\log n)$	$O(\log n)$

КОЛЕКЦИИ С НАРЕДБА VS КОЛЕКЦИИ БЕЗ НАРЕДБА

- TreeMap/TreeSet - червено-черни дървета. Запазват естествена наредба. Елементите трябва да имплементират интерфейса Comparable (или да се подава имплементация на Comparator). Логаритмична сложност за повечето операции.
- HashMap/HashSet - хеш таблици. Нямат естествена наредба. Елементите трябва да имплементират методите hashCode() и equals(). Константна сложност за повечето операции.


Java Map/Collection Cheat Sheet
Start
Will it contain key/value pairs or values only?
Pairs
Values
Is order important?
No
HashMap
Yes
Insertion order or sorted by keys?
Sorted
Ordered
TreeMap
LinkedHashMap
Will it contain duplicates?
Yes
ArrayList
No
Is primary task searching for elements (contains/remove)?
Yes
Is order important?
Yes
No
HashSet
Insertion order or sorted by values?
Ordered
LinkedHashSet
Sorted
TreeSet

КОЛЕКЦИИ - ОПЕРАЦИИ

- Сортиране

```
List<Integer> nums = new ArrayList<>();
nums.add(4);
nums.add(9);
nums.add(0);
nums.add(7);
nums.add(-1);

// nums = [4, 9, 0, 7, -1]
Collections.sort(nums);
// nums = [-1, 0, 4, 7, 9]
Collections.sort(nums, Collections.reverseOrder());
// nums = [9, 7, 4, 0, -1]
```

- Търсене: indexOf(), binarySearch()

```
List<Integer> nums = Arrays.asList(4, 9, 0, 7, -1);

// nums = [4, 9, 0, 7, -1]
int index = nums.indexOf(7);
// index = 3

Collections.sort(nums);
index = Collections.binarySearch(nums, -1);
// index = 0
```

- Разбъркване: shuffle()

```
List<Integer> nums = Arrays.asList(4, 9, 0, 7, -1);

// nums = [4, 9, 0, 7, -1]
Collections.shuffle(nums);
// nums = [?, ?, ?, ?, ?]
```

- Манипулаци **copy()**, **fill()**, reverse(), swap()

```
List<String> from = new ArrayList<>();
from.add("foo");
from.add("bar");
List<String> to = new LinkedList<>();
to.add("a");
to.add("b");

Collections.copy(to, from);
// to = [foo, bar]
```

```
List<String> list = new ArrayList<>();
list.add("foo");
list.add("bar");
list.add("baz");

Collections.fill(list, "a");
// list = [a, a, a]
```

- Манипулация copy(), fill(), **reverse()**, **swap()**)

```
List<String> list = new ArrayList<>();
list.add("1");
list.add("2");
list.add("3");

Collections.reverse(list);
// list = [3, 2, 1]

Collections.swap(list, 0, 1);
// list = [2, 3, 1]
```

- Статистики: min(), max(), frequency()

```
Set<Integer> nums = Set.of(4, 9, 0, 7, -1);

int min = Collections.min(nums); // -1
int max = Collections.max(nums); // 9
int frequency = Collections.frequency(nums, 7); // 1
```

Намиране на всички уникални думи от дадено множество:

```
import java.util.*;

public class FindDistinctWords {

    public static void main(String[] words) {
        Set<String> distinctWords = new TreeSet<>();
        for (String word : words) {
            distinctWords.add(word);
        }

        System.out.printf("%d distinct words: %s%n",
            distinctWords.size(), distinctWords);
    }
}
```

```
$ javac FindDistinctWords.java && \
  java FindDistinctWords "foo" "bar" "foo" "baz" "bar" "foo"
3 distinct words: [bar, baz, foo]
```

ПРЕМАХВАНЕ НА ЕЛЕМЕНТИ НА КОЛЕКЦИЯ ПРИ ИТЕРИРАНЕ

```
private static void filter(Collection<String> collection) {
    for (Iterator<String> it = collection.iterator();
         it.hasNext();) {
            if (it.next().charAt(1) == 'a') {
                it.remove();
            }
    }
}
```

```
Set<String> words = new HashSet<>();
words.add("foo");
words.add("bar");
words.add("baz");

filter(words);
// words = [foo]
```

COLLECTION FACTORY МЕТОДИ

```
List<String> list = List.of("Java", "9", "rulez");

Set<String> set = Set.of(1, 2, 3, 5, 8);

Map<String, Integer> cities = Map.of(
            "Brussels", 1_139_000,
            "Cardiff", 341_000
);

cities = Map.ofEntries(
            Map.entry("Brussels", 1_139_000),
            Map.entry("Cardiff", 341_000)
);
```

COLLECTION FACTORY МЕТОДИ

- Колекциите, създавани с factory методите, са immutable
- Заемат по-малко памет от mutable събратята си
- Не могат да съдържат null елементи
- При едно и също съдържание, могат да връщат нови инстанции или референции към съществуващи

ШАБЛОННИ ТИПОВЕ

ШАБЛОНИ (TEMPLATES)

- Клас или интерфейс, в декларацията на който има един или повече параметри за тип се нарича generic клас/интерфейс

```
List<E>
// We read "list of E"
```

- Дават възможност за параметризиране чрез типове на класове и методи
- Прави се проверка по време на компилация за съвместимост между типовете

НЕ-ШАБЛОННА КУТИЯ

Съдържа какъв да е обект

```
public class Box {
    private Object value;

    public Object getValue() {
        return value;
    }

    public void setValue(Object value) {
        this.value = value;
    }
}
```

ШАБЛОННА КУТИЯ

```
public class Box<T> {
    private T value;

    public T getValue() {
        return value;
    }

    public void setValue(T value) {
        this.value = value;
    }
}
```

СЪЗДАВАНЕ НА ИНСТАНЦИИ

```
// The long way - before Java 7
Box<Integer> integerBox = new Box<Integer>();
```

```
// Diamond operator - from Java 7
Box<Integer> integerBox = new Box<>();
```

Конвенция за именуване на параметрите за тип:

- E - Element
- T - Type
- K - Key
- V - Value
- N - Number
- S, U, V etc. - 2nd, 3rd, 4th types

ШАБЛОННИ МЕТОДИ

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

ШАБЛОННИ МЕТОДИ – ПРИМЕРИ

```
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 K getKey()   { return key; }
    public void setKey(K key) { this.key = key; }
    public V getValue() { return value; }
    public void setValue(V value) { this.value = 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());
    }
}
```

ШАБЛОННИ МЕТОДИ - ИЗВИКВАНЕ

```
Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");

// Full syntax
boolean areSame = Util.<Integer, String>compare(p1, p2);

// Short syntax
areSame = Util.compare(p1, p2);
```

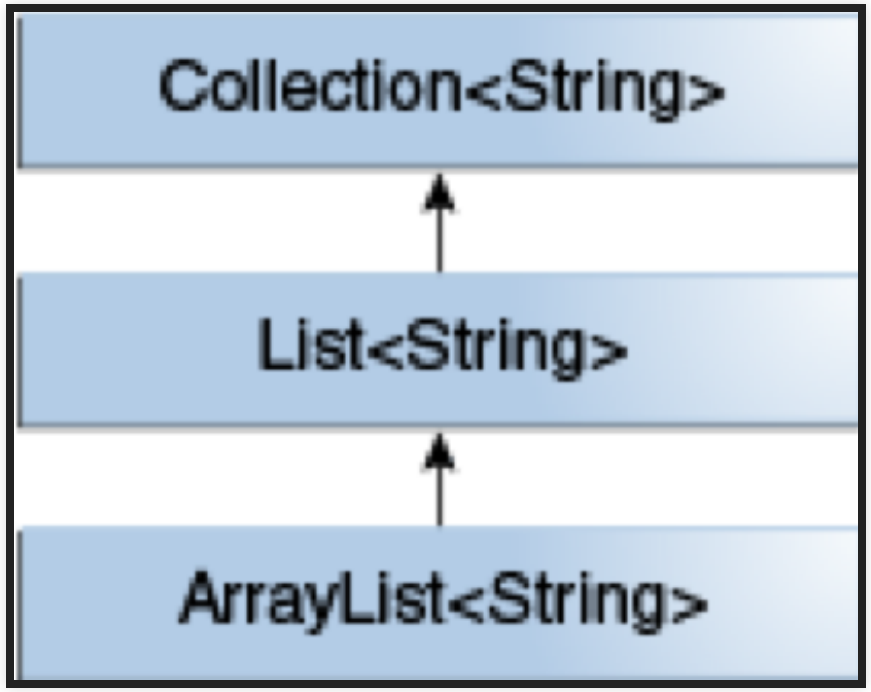
ШАБЛОННИ ТИПОВЕ - НАСЛЕДЯВАНЕ

- Integer is-a Object
- Integer is-a Number
- Double is-a Number
- Обаче `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 аргументът 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
- Не може да се използва за аргумент за тип при извикване на:
  - шаблонен метод
  - създаването на инстанция на шаблонен клас

WILDCARDS, ОГРАНИЧЕНИ ОТГОРЕ

Например, искате да създадете метод, който намира сумата на елементите в списък от Integer, Double, Float, Number.

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

WILDCARDS, ОГРАНИЧЕНИ ОТДОЛУ

- Искаме да създадем метод, които да добавя Integer обекти към списък
- Искаме методът да работи с колекции от Integer, Number, Object

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

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

ИЗТРИВАНЕ НА ИНФОРМАЦИЯТА ЗА ТИПОВИТЕ АРГУМЕНТИТЕ

- 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 или ограничени отдолу – с Object:

```
public class Box<T> {
    private T value;
    public T getValue() { return value; }
    public void setValue(T value) { this.value = value; }
}
```

- След заместване става:

```
public class Box {
    private Object value;
    public Object getValue() { return value; }
    public void setValue(Object value) { this.value = value; }
}
```

- Ако са ограничени отгоре – с техния ограничителен тип

```
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 типовия параметър от компилатора се получава:

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

СУРОВИ ТИПОВЕ (RAW TYPES)

- Представлява името на шаблонен клас или интерфейс без аргументите за тип
- Raw type на `Box<T>` е `Box`. Можем да създадем инстанция по следния начин

```
Box rawBox = new Box();
```

СУРОВИ ТИПОВЕ (RAW TYPES)

- 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:

```
// rawBox is a raw type of Box<T>
Box rawBox = new Box();

// warning: unchecked conversion
Box<Integer> intBox = rawBox;
```

СУРОВИ ТИПОВЕ (RAW TYPES)

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

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

// warning: unchecked invocation to set(T)
rawBox.setValue(8);
```

RAW TYPES И СЪВМЕСТИМОСТ НА ТИПОВЕТЕ

- `List` vs. `List<Object>`

```
List raw;
// warning: List is a raw type.
// References to generic type List<E> should be parameterized

List <Object> objects;
List <String> strings = new ArrayList<>();

raw = strings; // ?
objects = strings; // ?
```

ПРЕДПОЧИТАЙ LIST ВМЕСТО МАСИВ

```
// Fails at runtime
Object[] array = new Long[1];
array[0] = "I don't fit in"; // Throws ArrayStoreException

// Won't compile
List<Object> list = new ArrayList<Long>(); // Incompatible typ
list.add("I don't fit in");
```

ОГРАНИЧЕНИЯ ПРИ ШАБЛОННИТЕ КЛАСОВЕ И ИНТЕРФЕЙСИ

- Не могат 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);
}
```

- Не могат 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

- Не може да се правят конвертирания между типове (casts)
- Не може да се прилага instanceof операторът с шаблонни типове

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

ОГРАНИЧЕНИЯ ПРИ ШАБЛОННИТЕ КЛАСОВЕ И ИНТЕРФЕЙСИ

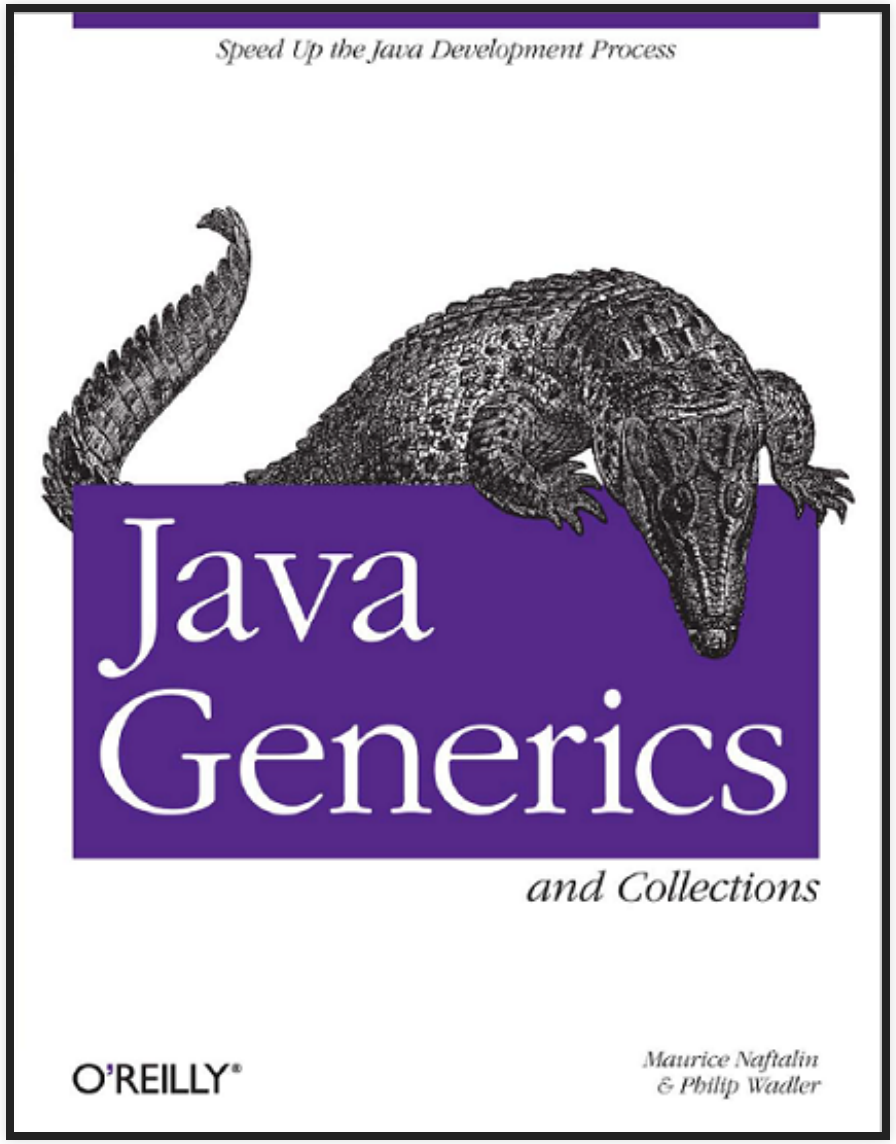
- Не може да се декларират масиви от параметризиран тип

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

// compiler error, but pretend it's allowed
Object[] stringLists = new List<String>[];

// OK
stringLists[0] = new ArrayList<String>();

// An ArrayStoreException should be thrown,
// but the runtime can't detect it
stringLists[1] = new ArrayList<Integer>();
```

ОГРАНИЧЕНИЯ ПРИ ШАБЛОННИТЕ КЛАСОВЕ И ИНТЕРФЕЙСИ

- Не може да се дефинират два метода с формални параметри, които след изтриване на типовите параметри имат еднакви сигнатури

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

    // compile-time error
    public void print(Set<Integer> intSet) { }
}
```


Speed Up the Java Development Process
Java
Generics
and Collections
O'REILLY®
Maurice Naftalin
& Philip Wadler

ВЪПРОСИ