Обобщени (шаблонни) типове (generics) в Java

Възможността да се работи с шаблонни типове в Java датира от JDK 5.0. Това, на практика, е въвеждане на оръжие за реализиране на алгоритмичен стил на програмиране в езика. При него типовете могат да бъдат абстрактни при деклариране и да се уточняват за конкретна употреба в процеса използването им. Освен икономия на писане, този стил подпомага избягването на грешки при ново и ново имплементиране на, по принцип, едни и същи идеи. Друга голяма полза от него е възможността да се поддържа по-голяма сигурност на типовете по време на работа.

Дефиниция

Обобщените (шаблонни) типове (generic types), понякога накратко наричани „шаблони“ (generics) са разширение на системата от типове в езика Java, което позволява създаването на класове с параметризирани типове. Така класовете могат да се превръщат в един вид „шаблони“, чиято типова конкретизация се извършва в момента на тяхното използване. Без тях сложни структури и алгоритми често се реализират наново и наново, което е трудоемко, може да води до грешки и намалява общността на създавани библиотеки. Ако пък тази общност се запази чрез използване на най-общия клас Object, както са реализациите преди Java 5.0, появява се непрестанна необходимост от промяна на типовете (тяхната конкретизация при всяко обръщение към методи на библиотеката), което, освен че бави и намалява четимостта, отново може да е източник на трудноотриваеми грешки, които се проявяват не по време на компилация, а по време на изпълнение.

Синтаксис

Създаване на класа

За дефиниране на клас с параметрични типове (шаблонен клас) се използват т. нар. „ъглови скоби“. Това са, всъщност, знаците за по-малко (<) и по-голямо (>), които в случая играят роля на „скоби“ – съответно отварящa и затварящa. Вътре в скобите има имена (обикновено започващи с главна буква, а най-често – само една главна буква), които при описването на класа по-надолу играят роля на „типове“. Те са разделени със запетая, ако са повече от едно. Ще реализираме за илюстрация един клас, който основно е предвиден да реализира метод swap, позволяващ размяна на стойностите на двете му свойства, условно наречени first и second. Естествено е този клас да бъде наречен на основната дейност, за която е реализиран – размяната.

Обърнете внимание: класът описва КАК да се извършва размяната на две стойности – т.е., **алгоритъма** чрез използване на трета променлива, познат до болка. И наистина, стойностите на променливи **от какъвто и да било тип** могат да бъдат разменяни така: създаваме временна променлива от същия тип, която да е със стойност, равна на първата променлива first; сега first е свободна да изгуби стойността си, приравнявайки се на втората (second); след което second може да получи съхранената стара стойност на first от временната променлива. Ето как ще изглежда класът, общо казано:

```
public class Swappable<T> {
 T first,second;
 public Swappable(T a, T b){
  first=a;
  second=b;
 }
 public T getFirst(){
  return first;
 }
 public T getSecond(){
  return second;
 }
 public void swap(){
  T t=first;
  first=second;
  second=t;
 }
```

```
 public String toString(){
  return "("+first+","+second+")";
 }
}
```

Използване на класа

Когато искаме да създадем обект от описания клас, в използвания конструктор вече обявяваме **същинските типове**, които да конкретизират параметрите на класа. Синтаксисът е подобен на описанието: при повикване на конструктора, след типа на променливата, се задават същинските типове, отново заградени в „ъглови скоби“ и разделени със запетая, ако са повече от един. Ето пример за използване на описания по-горе клас:

```
public class Test {
 public static void main(String[] args) {
  Swappable<Integer> t= new Swappable<Integer>(1,2);
  System.out.println(t.toString());
  t.swap();
  System.out.println(t.toString());
  Swappable<String> s=new Swappable<String>("КОТЕ","КОН");
  System.out.println(s.toString());
  s.swap();
  System.out.println(s.toString());
  Swappable<Fraction> f=new Swappable<Fraction>
                                    (new Fraction(1,2),new Fraction(3,4));
  System.out.println(f.toString());
  f.swap();
  System.out.println(f.toString());
 }
}
```

ВНИМАНИЕ! Простите типове в Java **не могат** да служат за конкретизация на шаблонни типове! Затова, обърнете внимание на реализацията по-горе, използваме генериращите типове: не ~~Swappable<int>~~, а Swappable<Integer>. Със String положението не е по-различно: просто класът String е специално разработен да изглежда като прост тип, но всъщност не е. String s="Hello"; и String s=new String("Hello"); са идентични декларации.