Обобщения(Generics), Collections

Обобщения - это параметризованные типы.

Параметризованные типы важны, поскольку позволяют объявлять классы, интерфейсы и методы, где тип данных, которыми они оперируют, указан в виде параметра. Используя обобщения, можно создать единственный класс, например, который будет автоматически работать с разными типами данных. Классы, интерфейсы или методы, имеющие дело с параметризованными типами, называются **обобщениями, обобщенными классами или обобщёнными методами.**

```
class Gen<T> {
  T ob; // declare an object of type T

  // Pass the constructor a reference to
  // an object of type T.
  Gen(T o) {
    ob = o;
  }

  // Return ob.
  T getob() {
    return ob;
  }

  // Show type of T.
  void showType() {
    System.out.println("Type of T is " +
                       ob.getClass().getName());
  }
}

// Demonstrate the generic class.
class GenDemo {
  public static void main(String args[]) {
    // Create a Gen reference for Integers.
    Gen<Integer> iOb;

    // Create a Gen<Integer> object and assign its
    // reference to iOb.  Notice the use of autoboxing
    // to encapsulate the value 88 within an Integer object.
    iOb = new Gen<Integer>(88);

    // Show the type of data used by iOb.
    iOb.showType();

    // Get the value in iOb. Notice that
    // no cast is needed.
    int v = iOb.getob();
    System.out.println("value: " + v);

    System.out.println();
```

Здесь T имя типа-параметра.

Обобщения работают только с объектами

```
Gen<int> strOb =new  Gen<int> (53) ; // Ошибка, нельзя использовать
// примитивные типы
```

Обобщенные типы отличаются в зависимости от типов-аргументов

iOb =strOb; // Не верно!

Несмотря на то, что iOb и strOb имеют тип Gen<T>, они являются ссылками на разные типы, потому что типы их параметров отличаются.

Обобщенный класс с двумя параметрами типа

```
// A simple generic class with two type
// parameters: T and V.
class TwoGen<T, V> {
  T ob1;
  V ob2;

  // Pass the constructor a reference to
  // an object of type T.
  TwoGen(T o1, V o2) {
    ob1 = o1;
    ob2 = o2;
  }

  // Show types of T and V.
  void showTypes() {
    System.out.println("Type of T is " +
                       ob1.getClass().getName());

    System.out.println("Type of V is " +
                       ob2.getClass().getName());
  }

  T getob1() {
    return ob1;
  }

  V getob2() {
    return ob2;
  }
}
/ Demonstrate TwoGen.
class SimpGen {
  public static void main(String args[]) {
```

Результат работы этой программы:
Тип T: j ava .lang. Integer
Тип V: java.lang.String
Значение: 88
Значение: Обобщения

Общая форма обобщенного класса

синтаксис объявления обобщённого класса:

class имя класса<список параметров типов> { // ...

Ниже показан синтаксис объявления ссылки на обобщенный класс:

имя класса<список аргументов типов> имя переменной = new имя класса <список аргументов типов>(список аргументов констант);

Ограниченные типы

В предыдущих примерах параметры типов могли быть заменены любыми типами классов. Это подходит ко многим случаям, но иногда удобно ограничить перечень типов, передаваемых в параметрах. Например, предположим, что вы хотите создать обобщенный класс, который содержит метод, возвращающий среднее значение массива чисел. Более того, вы хотите использовать этот класс для получения среднего значения чисел, включая целые и числа с плавающей точкой одинарной и двойной точности.

Чтобы справиться с этой ситуацией, java предлагает ограниченные типы. Когда указывается параметр типа, вы можете создать ограничение сверху, которое объявляет суперкласс, от которого все типы аргументы должны быть унаследованы.

```
class Gen <T extends superclass>
class Gen <T extends MyClass & MyInterface> ( // ...
```

```
class Stats<T extends Number> {
  T[] nums; // array of Number or subclass

  // Pass the constructor a reference to
  // an array of type Number or subclass.
  Stats(T[] o) {
    nums = o;
  }

  // Return type double in all cases.
  double average() {
    double sum = 0.0;

    for(int i=0; i < nums.length; i++)
      sum += nums[i].doubleValue();

    return sum / nums.length;
  }
}

// Demonstrate Stats.
class BoundsDemo {
  public static void main(String args[]) {

    Integer inums[] = { 1, 2, 3, 4, 5 };
    Stats<Integer> iob = new Stats<Integer>(inums);
    double v = iob.average();
    System.out.println("iob average is " + v);

    Double dnums[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
    Stats<Double> dob = new Stats<Double>(dnums);
    double w = dob.average();
    System.out.println("dob average is " + w);

    // This won't compile because String is not a
    // subclass of Number.
//    String strs[] = { "1", "2", "3", "4", "5" };
//    Stats<String> strob = new Stats<String>(strs);
```

class Gen<T extends MyClass & MyInterface> (1/ ... Здесь T ограничено классом по имени MyC1ass и интерфейсом MyInteface. То есть любой тип, переданный в T, должен быть подклассом MyClass и иметь реализацию MyInteface.

Использование шаблонных аргументов

```
Integer inums[] ={ 1, 2, 3, 4, 5 };
Double dnums[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };

Stats<Integer> iob = new Stats<Integer>(inums);
Stats<Double> dob  =new Stats<Double>(dnums);

if(iob.sameAvg(dob))
System.out.prinln("Средние значения одинаковы.");
else
System.out.println("Средние значения отличаются.");
```

Так как Stats параметризованный тип, какой тип параметра вы укажете для Stats, когда соз дадите параметр типа Stats?

!! Это не сработает!
// Определение эквивалентности двух средних значений.

```
boolean sameAvg(Stats<T> ob) {

if (average () ==ob.average())
return true;

}
```

```
//Определение эквивалентности двух средних значений.
// Отметим использования шаблонного символа.
// Определение эквивалентности двух средних значений.

boolean sameAvg(Stats<?> ob) {

if (average () ==ob.average())
return true;

}
```

Аргумент-шаблон специфицируется символом **?** и представляет собой неизвестный тип. Применение шаблона единственный способ написать работающий метод sameAvg ():

```
// Use a wildcard.
class Stats<T extends Number> {
  T[] nums; // array of Number or subclass

  // Pass the constructor a reference to
  // an array of type Number or subclass.
  Stats(T[] o) {
    nums = o;
  }

  // Return type double in all cases.
  double average() {
    double sum = 0.0;

    for(int i=0; i < nums.length; i++)
      sum += nums[i].doubleValue();

    return sum / nums.length;
  }

  // Determine if two averages are the same.
  // Notice the use of the wildcard.
  boolean sameAvg(Stats<?> ob) {
    if(average() == ob.average())
      return true;

    return false;
  }
}

// Demonstrate wildcard.
class WildcardDemo {
  public static void main(String args[]) {
    Integer inums[] = { 1, 2, 3, 4, 5 };
    Stats<Integer> iob = new Stats<Integer>(inums);
    double v = iob.average();
```

Ограниченные шаблоны

Аргументы шаблоны могут быть ограничены почти таким же способом, как параметры типов. Ограниченный шаблон особенно важен, когда вы создаете обобщенный тип, оперирующий иерархией классов.

```
/ Bounded Wildcard arguments.

// Two-dimensional coordinates.
class TwoD {
  int x, y;

  TwoD(int a, int b) {
    x = a;
    y = b;
  }
}

// Three-dimensional coordinates.
class ThreeD extends TwoD {
  int z;

  ThreeD(int a, int b, int c) {
    super(a, b);
    z = c;
  }
}

// Four-dimensional coordinates.
class FourD extends ThreeD {
  int t;

  FourD(int a, int b, int c, int d) {
    super(a, b, c);
    t = d;
  }
}

// This class holds an array of coordinate objects.
class Coords<T extends TwoD> {
  T[] coords;

  Coords(T[] o) { coords = o; }
}

// Demonstrate a bounded wildcard.
class BoundedWildcard {
  static void showXY(Coords<?> c) {
```

```
// Demonstrate a bounded wildcard.
class BoundedWildcard {
  static void showXY(Coords<?> c) {
    System.out.println("X Y Coordinates:");
    for(int i=0; i < c.coords.length; i++)
      System.out.println(c.coords[i].x + " " +
                         c.coords[i].y);
    System.out.println();
  }

  static void showXYZ(Coords<? extends ThreeD>
c) {
    System.out.println("X Y Z Coordinates:");
    for(int i=0; i < c.coords.length; i++)
      System.out.println(c.coords[i].x + " " +
                         c.coords[i].y + " " +
                         c.coords[i].z);
    System.out.println();
  }

  static void showAll(Coords<? extends FourD> c) {
    System.out.println("X Y Z T Coordinates:");
    for(int i=0; i < c.coords.length; i++)
      System.out.println(c.coords[i].x + " " +
                         c.coords[i].y + " " +
                         c.coords[i].z + " " +
                         c.coords[i].t);
    System.out.println();
  }

  public static void main(String args[]) {
    TwoD td[] = {
      new TwoD(0, 0),
      new TwoD(7, 9),
      new TwoD(18, 4),
      new TwoD(-1, -23)
    };

    Coords<TwoD> tdlocs = new Coords<TwoD>(td);

    System.out.println("Contents of tdlocs.");
    showXY(tdlocs); // OK, is a TwoD
```

В общем случае, для того, чтобы установить верхнюю границу шаблона, используйте следующий тип шаблонного выражения:

<? extends superclass>

здесь superclass это имя класса, который служит верхней границей. Класс, заданный в качестве верхней границы (т.е. superclass), также находится в пределах допустимых типов.
Вы также можете указать нижнюю границу шаблона, добавив выражение super к объявлению шаблона.

Вот его общая форма:

<? super subclass>

в этом случае допустимыми аргументами могут быть только классы, которые являются суперклассами для subclass. Это исключающая конструкция, поскольку она не включает класса subclass.

Создание обобщенного метода

```
// Демонстрация простого обобщенного метода.
class GenMethDemo {

// Определение, содержится ли объект в массиве.

static <T, V extends T> boolean isln (T x, V [] y) {

for(int i=0; i < y.length; i++)

if(x.equals(y[i])) return true;
return false;
}
```

Создание обобщенного метода

```
// Demonstrate a simple generic method.
class GenMethDemo {

  // Determine if an object is in an array.
  static <T, V extends T> boolean isIn(T x, V[] y) {

    for(int i=0; i < y.length; i++)
      if(x.equals(y[i])) return true;

    return false;
  }

  public static void main(String args[]) {

    // Use isIn() on Integers.
    Integer nums[] = { 1, 2, 3, 4, 5 };

    if(isIn(2, nums))
      System.out.println("2 is in nums");

    if(!isIn(7, nums))
      System.out.println("7 is not in nums");

    System.out.println();

    // Use isIn() on Strings.
    String strs[] = { "one", "two", "three",
                      "four", "five" };

    if(isIn("two", strs))
      System.out.println("two is in strs");

    if(!isIn("seven", strs))
```

синтаксис обобщенного метода:
<type-param-list> ret-type meth-name(param-list) (// ...

Результат работы этой программы показан ниже:
2 содержится в nums
7 не содержится в nums
два содержится в strs
семь не содержится в strs

Обобщенные конструкторы

Конструкторы также могут быть обобщенными, даже если их классы таковыми не являются.

```
// Use a generic constructor.
class GenCons {
  private double val;

  <T extends Number> GenCons(T arg) {
    val = arg.doubleValue();
  }

  void showval() {
    System.out.println("val: " + val);
  }
}

class GenConsDemo {
  public static void main(String args[]) {

    GenCons test = new GenCons(100);
    GenCons test2 = new GenCons(123.5F);

    test.showval();
    test2.showval();
  }
}
```

Обобщенные интерфейсы

В дополнение к обобщенным классам и методам вы можете объявлять обобщенные интерфейсы. Обобщенные интерфейсы специфицируются так же, как и обобщенные классы.

```
// A generic interface example.

// A Min/Max interface.
interface MinMax<T extends Comparable<T>> {
  T min();
  T max();
}

// Now, implement MinMax
class MyClass<T extends Comparable<T>> implements MinMax<T> {
  T[] vals;

  MyClass(T[] o) { vals = o; }

  // Return the minimum value in vals.
  public T min() {
    T v = vals[0];

    for(int i=1; i < vals.length; i++)
      if(vals[i].compareTo(v) < 0) v = vals[i];

    return v;
  }

  // Return the maximum value in vals.
  public T max() {
    T v = vals[0];

    for(int i=1; i < vals.length; i++)
      if(vals[i].compareTo(v) > 0) v = vals[i];

    return v;
  }
```

class имякласса<список параметров типов>
implements имя интерфейса<список аргументовтипов>

Иерархии обобщенных классов

Обобщенные классы могут быть частью иерархии классов так же, как и любые другие необобщенные классы. То есть обобщенный класс может выступать в качестве суперкласса или подкласса. Ключевое отличие между обобщенными и необобщенными иерархиями состоит в том, что в обобщенной иерархии любые аргументы типов, необходимые обобщенному суперклассу, всеми подклассами должны передаваться по иерархии вверх.

Использование обобщенного суперкласса

```
// A simple generic class hierarchy.
class Gen<T> {
  T ob;

  Gen(T o) {
    ob = o;
  }

  // Return ob.
  T getob() {
    return ob;
  }
}

// A subclass of Gen.
class Gen2<T> extends Gen<T> {
  Gen2(T o) {
    super(o);
  }
}
```

```
class Gen<T> {
  T ob;

  Gen(T o) {
    ob = o;
  }

  // Return ob.
  T getob() {
    return ob;
  }
}

// A subclass of Gen.
class Gen2<T> extends Gen<T> {
  Gen2(T o) {
    super(o);
  }
}

// Demonstrate run-time type ID implications of generic
// class hierarchy.
class HierDemo3 {
  public static void main(String args[]) {

    // Create a Gen object for Integers.
    Gen<Integer> iOb = new Gen<Integer>(88);

    // Create a Gen2 object for Integers.
    Gen2<Integer> iOb2 = new Gen2<Integer>(99);

    // Create a Gen2 object for Strings.
    Gen2<String> strOb2 = new Gen2<String>("Generics Test");

    // See if iOb2 is some form of Gen2.
    if(iOb2 instanceof Gen2<?>)
      System.out.println("iOb2 is instance of Gen2");

    // See if iOb2 is some form of Gen.
    if(iOb2 instanceof Gen<?>)
      System.out.println("iOb2 is instance of Gen");
```

Обобщенный подкласс

Абсолютно приемлемо, когда суперклассом для обобщенного класса выступает класс необобщенный.

Нельзя создавать экземпляр типа пара метра

Создавать экземпляр типа параметра невозможно.
Например, рассмотрим такой класс:

```
// Нельзя создавать экземпляр типа T.
class Gen<T>
T ob;

Gen () {
 ob= new T(); // Недопустимо!!!
}
```

Ограничения на статические члены

Никакой static член не может использовать тип параметра, объявленный в его классе. Например, все static члены этого класса являются недопустимыми:

```
class Wrong<T> {
// Неверно, нельзя создать статические переменные типа T.
static T ob;

// Неверно, ни один статический метод не может использовать T.
static T getob() {
return ob;
}

// Неверно, ни один статический метод не может иметь доступ
// к объекту типа T.

static void showob() {
System.out.println(ob);
```

Ограничения обобщенных массивов

```
// Обобщения и массивы.

class Gen<T extends Number> {
т ob;
т vals[]; // ОК
Gen{T o, T[] nums)
ob o;
// Этот оператор неверен.
// vals = new T[10]; // нельзя создавать массив объектов Т
// Однако этот оператор верен.
vals = nums; // можно присвоить ссылку существующему массиву
}}
class GenArrays {
public static void main(String args[])
Integer n[] = { 1, 2, 3, 4, 5 );
Gen<Integer> iOb = new Gen<Integer> (50, n);
// Нельзя создать массив специфичных для типа обобщенных ссылок.
// Gen<Integer> gens[] = new Gen<Integer>[10]; // Неверно!
/ / Это верно.
Gen<?> gens[] = new Gen<?>[10]; // ОК

}
}
```

Обобщения это мощное расширение языка java, поскольку они упрощают создание безопасного в отношении типов и повторно используемого кода. Хотя обобщенный синтаксис поначалу может показаться несколько громоздким, обобщенный код станет частью будущего для всех программистов на java.