Java — Дженерики (обобщения)

Было бы неплохо, если бы мы могли написать один метод сортировки, который мог бы сортировать элементы в массиве Integer, массиве String или массиве любого типа, поддерживающего упорядочение.

Java методы обобщения и общие классы позволяют программистам указывать с объявлением одного метода набор связанных методов или с объявлением одного класса набор связанных типов, соответственно.

В Java дженерики также обеспечивают безопасность типа компиляции, которая позволяет программистам ловить недопустимые типы во время компиляции.

Используя концепцию Java Generic, мы можем написать общий метод для сортировки массива объектов, а затем вызвать общий метод с массивами Integer, Double, String и т.д., чтобы отсортировать элементы массива.

Общие методы

Вы можете написать одно обобщенное объявление метода, которое можно вызвать с помощью аргументов разных типов. Основываясь на типах аргументов, переданных на общий метод, компилятор обрабатывает каждый вызов метода соответствующим образом. Ниже приведены правила определения общих методов:

- Все объявления обобщенного метода имеют раздел параметров типа, разделенный угловыми скобками (), который предшествует возвращаемому типу метода (в следующем примере).
- Раздел параметров каждого типа содержит один или несколько параметров типа, разделенных запятыми. Параметр типа, также известный как переменная типа, является идентификатором, который указывает общее имя типа.
- Параметры типа могут использоваться для объявления возвращаемого типа и действуют как заполнители для типов аргументов, переданных в общий метод, которые известны как аргументы фактического типа.
- Тело общего метода объявлено как тело любого другого метода. Обратите внимание, что параметры типа могут представлять только ссылочные типы, а не примитивные типы (например, int, double и char).

Пример

Следующий пример показывает, как мы можем выводить массив различного типа, используя один общий:

```
public class GenericMethodTest {
   // Общий метод printArray
   public static < E > void printArray( E[] inputArray ) {
      // Отображаем элементы массива
      for(E element : inputArray) {
         System.out.printf("%s ", element);
      }
      System.out.println();
   }

   public static void main(String args[]) {
      // Создание массивов типа Integer, Double и Character
      Integer[] intArray = { 1, 2, 3, 4, 5 };
      Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
      Character[] charArray = { 'П', 'Р', 'И', 'В', 'Е', 'Т'};

      System.out.println("Массив integerArray содержит:");
      printArray(intArray);   // передать массив Integer

      System.out.println("\nМассив doubleArray содержит:");
      printArray(doubleArray);   // передать массив Double

      System.out.println("\nМассив characterArray содержит:");
      printArray(charArray);   // передать массив Character
   }
}
```

Получим следующий результат:

```
Массив integerArray содержит:
1 2 3 4 5

Массив doubleArray содержит:
1.1 2.2 3.3 4.4

Массив characterArray содержит:
П Р И В Е Т
```

Параметры ограниченного типа

Могут быть случаи, когда вы захотите ограничить виды типов, которым разрешено быть переданными типу параметра. Например, метод, который работает с числами, может только принимать экземпляры Number или его подклассов. Для этого используются параметры ограниченного типа.

Чтобы объявить параметр ограниченного типа, введите имя параметра типа, за которым следует ключевое слово extends, а затем его верхняя граница.

Пример

Следующий пример иллюстрирует, как extends используется в общем смысле для обозначения либо «extends» (как в классах), либо «implements» (как в интерфейсах). Это пример общего метода для возврата самого большого из трех объектов Comparable:

```
public class MaximumTest {
   // Определяем наибольший из трёх Comparable объектов

   public static > T maximum(T x, T y, T z) {
      T max = x;   // Предположим, что x изначально максимальный

      if(y.compareTo(max) > 0) {
         max = y;   // здесь y пока что наибольший
      }

      if(z.compareTo(max) > 0) {
         max = z;   // здесь z наибольший сейчас
      }
      return max;   // возвращается максимальный объект
   }

   public static void main(String args[]) {
      System.out.printf("Максимумом из %d, %d и %d является %d\n\n",
         3, 4, 5, maximum( 3, 4, 5 ));

      System.out.printf("Максимумом из %.1f,%.1f и %.1f является %.1f\n\n",
         6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ));

      System.out.printf("Максимумом из %s, %s и %s является %s\n","груша",
        "яблоко", "апельсин", maximum("груша", "яблоко", "апельсин"));
   }
}
```

Получим следующий результат:

```
Максимумом из 3, 4 и 5 является 5

Максимумом из 6.6,8.8 и 7.7 является 8.8

Максимумом из груша, яблоко и апельсин является груша
```

Общие классы

Объявление общего класса выглядит как объявление не общего класса, за исключением того, что за имен типа.

Как и в случае с общими методами, раздел параметров типа универсального класса может иметь один ил разделенных запятыми. Эти классы известны как параметризованные классы или параметризованные т несколько параметров.

Пример

Следующий пример показывает, как мы можем определить общий класс:

```
public class Box {
   private T t;

   public void add(T t) {
      this.t = t;
   }

   public T get() {
      return t;
   }

   public static void main(String[] args) {
      Box integerBox = new Box();
      Box stringBox = new Box();

      integerBox.add(new Integer(10));
      stringBox.add(new String("Привет Мир"));

      System.out.printf("Значение Integer :%d\n\n", integerBox.get());
      System.out.printf("Значение String :%s\n", stringBox.get());
   }
}
```

Получим следующий результат:

```
Значение Integer :10
Значение String :Hello World
```