Модуль 2. Введение в объектно-ориентированное программирование

2.8. * Параметризованные типы

Понятие обобщенного класса

IT Школа

Обобщенное программирование (generic programming) — это подход к описанию данных и алгоритмов, который предусматривает их использование с различными типами данных без изменения их описания.
Дженерики — это параметризованные типы. С помощью параметризованных типов можно объявлять классы, интерфейсы и методы, при этом тип данных выступает в виде параметра.

К основным **свойствам** дженериков можно отнести:
- строгую типизацию;
- единую реализацию;
- отсутствие информации о типе.

```
class name<T1, T2, ..., Tn> { /* ... */ }
```

- **E** — элемент (широко используется в Java Collection Framework);
- **K** — ключ;
- **N** — число;
- **T** — тип;
- **V** — значение;
- **S, U, V** и т. д. — 2-й, 3-й, 4-й типы.

Пример

IT Школа

```java
class Generic<T>{
        T t; // объявляем объект типа T

        //передаем в конструктор параметр типа T
        public Generic(T t) {
                this.t = t;
        }

        //получаем параметр типа T
        public T getT() {
                return t;
        }

        //метод вывода параметризованного типа на экран
        public void print(){
                System.out.println("Value T: " + getT());
                System.out.println("Type T: " + t.getClass().getName());
        }
}
```

```java
Generic<Integer> integerGeneric = new Generic<>(new Integer(5));
integerGeneric.print();

System.out.println();

Generic<String> stringGeneric = new Generic<>("Hello");
stringGeneric.print();
```

```
Value T: 5 Type T: java.lang.Integer
Value T: Hello Type T: java.lang.String
```

```
class Generic<T extends Number>{}
```

Обобщенный класс с несколькими параметрами

IT Школа

```
class Pair<K, V>{
        K k;
        V v;

        //Передаем в конструктор ссылки на K и V
        public Pair(K k, V v) {
                super();
                this.k = k;
                this.v = v;
        }

        public void print(){
                System.out.println("Value K: " + getK());
                System.out.println("Type K: " + k.getClass().getName());
                System.out.println("Value V: " + getV());
                System.out.println("Type V: " + v.getClass().getName());
        }

        public K getK() {
                return k;
        }

        public V getV() {
                return v;
        }
}
```

```
Pair<String, Integer> pair = new Pair <> ("Hello", new Integer(5));
pair.print();
```

```
Value K: Hello Type K: java.lang.String Value V: 5 Type V: java.lang.Integer
```

```
Pair<String, Integer> pair = new Pair<String, Integer>("Hello", new Integer(5)); //до JDK 7
Pair<String, Integer> pair = new Pair<>("Hello", new Integer(5)); //начиная с JDK 7
```

Универсальные методы (generic methods)

IT Школа

```
public void printArr(Integer [] iArr){
        for(int i = 0; i < iArr.length; i++){
                System.out.print(iArr[i] + " ");
        }
        System.out.println();
}

public void printArr(Double [] dArr){
        for(int i = 0; i < dArr.length; i++){
                System.out.print(dArr[i] + " ");
        }
        System.out.println();
}

public void printArr(Boolean [] bArr){
        for(int i = 0; i < bArr.length; i++){
                System.out.print(bArr[i] + " ");
        }
        System.out.println();
}

public void printArr(Character [] cArr){
        for(int i = 0; i < cArr.length; i++){
                System.out.print(cArr[i] + " ");
        }
        System.out.println();
}
```

```
public <T> void printArr(T [] tArr){
        for(int i = 0; i < tArr.length; i++){
                System.out.print(tArr[i] + " ");
        }
        System.out.println();
}
```

```
Integer [] iArr = {1,2,3,4,5};
Double [] dArr = {1.0, 2.0, 3.0, 4.0, 5.0};
String [] sArr = {"a", "b", "c", "d", "e"};
printArr(iArr);
printArr(dArr);
printArr(sArr);
```

```
1 2 3 4 5 1.0 2.0 3.0 4.0 5.0 a b c d e
```

Методы с параметризованными типами

IT Школа

```
<T> void method(Generic<T> generic){}
```

```
void method(Generic<?> generic){}
```

- **? extends T** — определяет множество классов потомков T;
- **? super T** — определяет множество родительских классов T.

```
public void print(Generic<? extends Number> generic){
    generic.print();
}
```

Спасибо за внимание!