Практическая работа №9

Теория по дженерикам

Дженерики (generics – в пер. с англ. «обобщения») позволяют обнаруживать ошибки уже во время компиляции программы, а не во время ее выполнения.

Дженерики позволяют параметризовать типы, т.е. сделать типы зависимыми от параметров. С помощью этой возможности можно определить класс или метод с обобщенными типами, которые компилятор может заменить на конкретные. Например, Java определяет обобщенный класс ArrayList для хранения элементов обобщенного типа. Из этого обобщенного класса можно создать объект типа ArrayList для хранения строк и объект типа ArrayList для хранения чисел. Здесь строки и числа являются конкретными типами, на которые заменяется обобщенный.

Java позволяет определять обобщенные классы, интерфейсы и методы, начиная с JDK 1.5. Несколько интерфейсов и классов в Java API были изменены с помощью дженериков. Например, до JDK 1.5 интерфейс java.lang.Comparable был определен, как показано слева, а начиная с JDK 1.5 он был изменен, как показано справа.

До JDK 1.5	JDK 1.5

package java.lang;
public interface Comparable {
public int compareTo(Object o)
}
	package java.lang;
	public interface Comparable {
	public int compareTo(T o)
	}

Здесь представляет формальный обобщенный тип, который можно позже заменить на фактический конкретный тип. Замена обобщенного типа называется обобщенным инстанцированием. В соответствии с соглашением, для обозначения формального обобщенного типа используется заглавная буква, такая как E или T.

Задание №1

1. Напишите метод, которому передается коллекция объектов типа ArrayList, а возвращается коллекция ArrayList, но уже без дубликатов. Необходимо использовать метод contains() интерфейса List.
2. Реализуйте алгоритм линейного поиска элемента в массиве. При нахождении элемента необходимо вернуть его позицию в массиве. Если элемент не найден, то

вернуть -1.

3. Реализуйте поиск наибольшего элемента в массиве с помощью метода compareTo() интерфейса Comparable. Определите класс Circle с полем radius и найдите наибольший элемент в массиве экземпляров этого класса.
4. Реализуйте поиск наибольшего элемента в двумерном массиве с помощью метода compareTo() интерфейса Comparable.

Задание №2

1. Измените класс GenericStack таким образом, чтобы реализовать его с помощью массива, а не ArrayList. Перед добавлением нового элемента в стек необходимо проверить размер массива. Если массив заполнен, создайте новый массив, который удвоит текущий размер массива и скопирует элементы из текущего массива в новый.
2. Класс GenericStack из предыдущего описания реализован с помощью отношения композиции. Определите новый класс стека, который наследуется от ArrayList. Нарисуйте UML-диаграмму этих классов, а затем реализуйте новый класс GenericStack. Напишите тестовую программу, которая запросит у пользователя пять строк, а отобразит их в обратном порядке.

GenericStack < E >	
-list java.util.ArrayList	Список для хранения элементов
+GenericStack()	Создает пустой стек
+getSize(): int	Возвращает количество элементов в этом стеке
+peek(): E	Возвращает элемент на вершине этого стека, не удаляет его
+poop(): E	Возвращает и удаляет элемент на вершине этого стека
+push(o: E): void	Добавляет новый элемент на вершину этого стека
+isEmpty(): boolean	Возвращает значение true, если этот стек пустой

```
public class GenericStack<E> {
    private java.util.ArrayList<E> list = new java.util.ArrayList<>();

    public int getSize() {
        return list.size();
    }

    public E peek() {
        return list.get(getSize() - 1);
```

```
    }

    public void push(E o) {
        list.add(o);
    }

    public E pop() {
        E o = list.get(getSize() - 1);
        list.remove(getSize() - 1);
        return o;
    }

    public boolean isEmpty() {
        return list.isEmpty();
    }

    @Override
    public String toString() {
        return "стек: " + list.toString();
    }
}
```