Generic Types

by Vlad Costel Ungureanu

for Learn Stuff IO

LEARN stuff IO ACADEMY

Why Generics Types ?

- ✓Generic class declarations
- ✓Generic interface declarations
- ✓Generic method declarations
- ✓Generic constructor declarations

LEARN stuff IO ACADEMY

Generic Types

✓Applied to Classes:

public class ClassName<T1, T2, ..., Tn> { … }

✓ Class example:

```
public class CustomList<E> {
    // E represents the generic data type
    private E[] items;
    public void push(E item) { ... }
    public E peek() { .. }
}
```

✓Applied to Interfaces:

public interface InterfaceName<T1, T2, ..., Tn> { … }

LEARN stuff IO ACADEMY

Generic Types

✓ Parametrize data types by using data types as parameters

public CustomList<String> myList = new CustomList<String>();

✓ Enhanced control over the way types are used

✓ We **cannot** add any other type to the custom list, thus **achieving type safety**

~~myList.add(new Rectangle());~~

✓ Type checking (**safety**) is done at compile time (**cast is no longer required**)

String s = ~~(String)~~ myList.peek();

LEARN stuff IO ACADEMY

Generic Types – Type Letter Conventions

- ✓ E – Element (intensive use in Java Collections)
- ✓ K - Key
- ✓ N - Number
- ✓ T - Type
- ✓ V - Value
- ✓ S,U,V etc. - 2nd, 3rd, 4th types

- public class Node<T> { … }
- public interface Pair<K, V> { … }
- public class PairImplementation<K, V> implements Pair<K, V> {...}

LEARN stuff IO ACADEMY

Generic Types - Usage

✓ Generic invocation of types:

```
CustomList<String> stack = new CustomList<String>();
Pair<Integer,String> pair = new PairImpl<Integer,String>(0, "ab");
Stack<Node<Integer>> nodes = new Stack<Node<Integer>>();
```

✓ Diamond operator “<>” (for reduce verbosity):

```
CustomList<String> stack = new CustomList<>();
Pair<Integer,String> pair = new PairImpl<>(0, "ab");
Stack<Node<Integer>> nodes = new Stack<>();
```

✓ The diamond operator is used for **type inference**

✓ Type inference refers to the fact that the compiler determines the most suitable constructor declaration that matches the invocation in our code

LEARN stuff IO ACADEMY

Generic Types – Generic Methods

```
public class Util {
        public static <T> int countNullValues(T[] anArray) {
        int count = 0;
        for (T e : anArray) {
                if (e == null) {
                        ++count;
                }
        }
        return count;
        }
}
```

- Util.countNullValues(new String[]{"a", null, "b"});
- Util.countNullValues(new Integer[]{1, 2, null, 3, null});

LEARN stuff IO ACADEMY

Generic Types – Bounded Types

✓ **Superior Bound**

```
public double sumOfList(List<? extends Number> list) {
        double s = 0.0;
        for (Number n : list) { s += n.doubleValue(); }
        return s;
}
```

✓ **No Bounds**

```
public void printList(List<?> list) {
        for (Object elem: list) { System.out.print(elem + " "); }
}
```

✓ **Inferior Bound**

```
public void addNumbers(List<? super Integer> list) {
 for (int i = 1; i <= 10; i++) {
         list.add(i); }
}
```

LEARN stuff IO ACADEMY

Generic Types – Bounded Types

```
public class Util {
        // generic method
        public <U extends Integer> void inspect(U u){
                System.out.println("U: " + u.getClass().getName());
        }
        public static void main(String[] args) {
                Node<Double> node = new Node<>();
                node.inspect(12.34); // correct
                node.inspect(1234); // correct
                node.inspect("some text"); // not allowed
        }
}
```

LEARN stuff IO ACADEMY

Assignments

- ✓ Write a class that simulates a stack and a FIFO (first in first out) queue using generics and arrays
- ✓ For the previous classes add the following operations: add, remove, get, list and isEmpty
- ✓ Using generics to describe an object dictionary. This class should contain a list of key-value pairs where the key is a String and the value a generic typed object.

LEARN stuff IO ACADEMY

THANK YOU!

Vlad Costel Ungureanu

contact@learnstuffacademy.io

This is a free course from LearnStuff.IO

– not for commercial use –