



Generic Types

by Vlad Costel Ungureanu
for "Learn Stuff"

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

- ✓ `class ClassName<T1, T2, ..., Tn> { ... }`
- ✓ `interface InterfaceName<T1, T2, ..., Tn> { ... }`

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

- ✓ Parametrize data types by using data types as parameters

```
public CustomList<String> myList ...
```

- ✓ Enhanced control over the way types are used

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

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

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

- ✓ 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> {...}`

- ✓ 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 "<>":

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

```
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});

```
private T t;
public void set(T t) { this.t = t; }
public T get() { return t; }
// Metoda generica
public <U extends Integer> void inspect(U u){
    System.out.println("T: " + t.getClass().getName());
    System.out.println("U: " + u.getClass().getName());
}
public static void main(String[] args) {
    Node<Double> node = new Node<>();
    node.set(12.34); //OK
    node.inspect(1234); //OK
node.inspect(12.34);
node.inspect("some text");
}
}
```


✓ 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); }  
}
```

- ✓ Write a class that simulate 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 a 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.

THANK YOU!

Vlad Costel Ungureanu
ungureanu_vlad_costel@yahoo.com

This is a free course from [LearnStuff.io](https://learnstuff.io)
– not for commercial use –