



Java Generics

Αθ. Ανδρούτσος



Γενικές μέθοδοι και κλάσεις

Προγραμματισμός με Java

- Πολλές φορές χρειάζεται να ορίζουμε interfaces, κλάσεις ή μεθόδους που να επιτρέπουν ο κώδικάς τους να μπορεί να χρησιμοποιηθεί από περισσότερους από έναν τύπους δεδομένων με την **ίδια λειτουργικότητα**
- Πρόκειται ουσιαστικά για μηχανισμό **επαναχρησιμοποίησης** του κώδικα



Παράδειγμα - Κλάση Node

Προγραμματισμός με Java

```
1  package gr.aueb.cf.ch19.generics;
2
3  /**
4   * Ορίζει μία κλάση {@link Node} με
5   * ένα μόνο πεδίο int.
6   */
7  public class Node {
8      private int item;
9
10     public int getItem() {
11         return item;
12     }
13
14     public void setItem(int item) {
15         this.item = item;
16     }
17 }
```

- Η κλάση **Node** αναπαριστά ένα κόμβο που περιλαμβάνει ένα πεδίο **item** τύπου **int**



Κλάση NodeDemo

Προγραμματισμός με Java

```
1 package gr.aueb.cf.ch19.generics;
2
3 public class NodeApp {
4
5     public static void main(String[] args) {
6         Node node = new Node();
7
8         node.setItem(7);
9         int data = node.getItem();
10
11         System.out.println(data);
12     }
13 }
```

- Σε μια driver class κάνουμε set και get μια απλή τιμή int

Μπορεί ωστόσο να θέλουμε να χρησιμοποιήσουμε τη λειτουργικότητα του Node και για άλλους τύπους δεδομένων εκτός από int.



Node Class

Προγραμματισμός με Java

- Η κλάση Node είναι περιοριστική και καθόλου γενική
- Αν θέλαμε να έχουμε nodes που να περιέχουν String, Float, Double, κλπ. πρέπει να φτιάχνουμε μία κλάση για κάθε διαφορετικό τύπο
- Ίσως μια λύση θα ήταν η κλάση μας αντί για int να **περιέχει Object**, οπότε τότε θα μπορούσαμε να δημιουργούμε instances που να περιέχουν οποιοδήποτε τύπο μια μιας και **όλοι οι τύποι IS-A Object**



Flexible Node

Προγραμματισμός με Java

```
1 package gr.aueb.cf.ch19.generics;
2
3 /**
4  * Ορίζει ένα γενικό τύπο Node, όπου
5  * στη θέση του Object μπορούμε να έχουμε
6  * οποιοδήποτε τύπο δεδομένων.
7  */
8 public class NodeFlexible {
9     private Object item;
10
11     public Object getItem() { return item; }
12
13     public void setItem(Object item) { this.item = item; }
14
15 }
16
17 }
```

- Η κλάση **NodeFlexible** αναπαριστά ένα κόμβο που περιλαμβάνει ένα πεδίο **item** τύπου **Object**

Το γεγονός ότι το πεδίο **item** είναι **Object** προσδίδει ευελιξία αφού μπορούμε να χρησιμοποιήσουμε ως **item** οποιονδήποτε τύπο



NodeFlexible Driver Class

Προγραμματισμός με Java

```
1 package gr.aueb.cf.ch19.generics;
2
3 public class NodeFlexibleApp {
4
5     public static void main(String[] args) {
6         NodeFlexible head = new NodeFlexible();
7         head.setItem("CF");
8         int data = (int) head.getItem();
9
10        System.out.println(data);
11    }
12 }
```

- Το πρόβλημα είναι ότι όταν στην γραμμή 8, πάμε να κάνουμε cast από String σε int, ο μεταγλωττιστής **δεν βγάζει checked exception** αφού δεν μπορούμε να μετατρέψουμε από String σε int, αλλά μόνο όταν το τρέχουμε δίνει *ClassCastException (runtime exception)*

Run: NodeFlexibleApp x

```
"C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe" "-javaagent:C:\Pro
Exception in thread "main" java.lang.ClassCastException: class java
    at gr.aueb.cf.ch19.generics.NodeFlexibleApp.main(NodeFlexibleApp.java:8)
Process finished with exit code 1
```

Επομένως, δεν έχουμε compile-type safety χρησιμοποιώντας **Object** ως τύπο του πεδίου της **Node**



Γενικές κλάσεις - Ορισμός

Προγραμματισμός με Java

- Μία γενική κλάση ή interface είναι ένας ειδικός τύπος κλάσης ή interface που ορίζει ένα ή περισσότερους μη-συγκεκριμένους (γενικούς) τύπους κατά τον ορισμό της
- Γενικούς τύπους ορίζουμε μέσα σε angle brackets `<>`, π.χ. `<E>` όπου E είναι ένα γενικός τύπος



Δήλωση

Προγραμματισμός με Java

- Κατά τη δήλωση όμως δηλώνουμε συγκεκριμένους τύπους, οι οποίοι μπορεί να είναι **μόνο Reference Types**
- Έτσι δεν έχουμε `ClassCastException` όταν μετατρέπουμε μεταξύ διαφορετικών τύπων, γιατί οποιοδήποτε πρόβλημα έχει εντοπιστεί ήδη κατά τη μεταγλώττιση



Γενική κλάση NodeGeneric

Προγραμματισμός με Java

```
1 package gr.aueb.cf.ch19.generics;
2
3 public class NodeGeneric<T> {
4     private T item;
5
6     public T getItem() {
7         return item;
8     }
9     public void setItem(T item) {
10         this.item = item;
11     }
12
13     @Override
14     public String toString() {
15         return "NodeGeneric{" +
16             "item=" + item +
17             '}';
18     }
19 }
```

- Παράμετροι τύπων (<T> από το Type)
- Οι γενικές κλάσεις δηλώνουν ένα ή περισσότερους παραμετρικούς τύπους
- Το όνομα του τύπου είναι NodeGeneric<T>. Τον παραμετρικό τύπο T μπορούμε και τον χρησιμοποιούμε μέσα στην κλάση για να δηλώνουμε τύπους πεδίων και μεθόδων είτε επιστρεφόμενων τιμών ή παραμέτρων



Χρήση αντικειμένων γενικών τύπων

Προγραμματισμός με Java

```
1 package gr.aueb.cf.ch19.generics;
2
3 public class NodeGenericApp {
4
5     public static void main(String[] args) {
6         NodeGeneric<Integer> intNode = new NodeGeneric<>();
7         NodeGeneric<String> strNode = new NodeGeneric<>();
8
9         intNode.setItem(1);
10        strNode.setItem("CF");
11
12        System.out.println(intNode);
13        System.out.println(strNode);
14    }
15 }
```

- Η δήλωση γενικών τύπων γίνεται μέσα σε **<>**, π.χ. **<Integer>** ενώ ο παραμετρικός τύπος **πρέπει να είναι κλάση** και όχι πρωταρχικός τύπος
- Ο Constructor ορίζεται επίσης με το **<>** χωρίς τύπο μέσα στα **<>** αφού **ο τύπος συμπεραίνεται** από τον τύπο της δήλωσης στο αριστερό μέρος

Στο παράδειγμα, δηλώνοντας **<Integer>** και **<String>** ως τον παραμετρικό τύπο των στοιχείων της *NodeGeneric* το πρόβλημα με την εκχώρηση **String** σε **Integer** εντοπίζεται από τον μεταγλωττιστή, κι έτσι δεν δημιουργείται run-time error (δηλ. unchecked exception)



Δήλωση αντικειμένων γενικών τύπων

Προγραμματισμός με Java

```
NodeGeneric<Integer> intNode = new NodeGeneric<>();  
NodeGeneric<String> strNode = new NodeGeneric<>();
```

- Η μόνη διαφορά σε σχέση με τη **δήλωση** αντικειμένων απλών τύπων είναι ότι στους γενικούς τύπους προσθέτουμε τον παραμετρικό τύπο `<Integer>` , `<String>` και γενικά τον συγκεκριμένο τύπο μέσα στον τελεστή `<>`



Java 10

```
1 package testbed.ch19;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         var intNode = new Node<Integer>();
7         var strNode = new Node<String>();
8
9         intNode.setItem(1);
10        strNode.setItem("a");
11
12        System.out.println(intNode.getItem());
13        System.out.println(strNode.getItem());
14    }
15 }
```

- Στην Java 10 και μετά μπορούμε να δηλώνουμε τοπικές μεταβλητές με το keyword **var** όταν ο τύπος μπορεί να γίνει infer (να τον συμπεράνει η Java) από το δεξί μέρος της παράστασης



Generics

- Τα **Generics** επιτρέπουν σε κλάσεις, μεθόδους και interfaces να λειτουργούν με διάφορους τύπους δεδομένων, δηλώνοντας τον τύπο μέσα σε `<>` και παρέχοντας έτσι *compile-time safety*, δηλαδή υλοποιούν τη βασική αρχή *“eliminate every unchecked warning that you can”*
- Πέρα από αυτό, τα Generics είναι επίσης μηχανισμός επαναχρησιμοποίησης του κώδικα



Generics με δύο τύπους (1)

Προγραμματισμός με Java

```
1 package gr.aueb.cf.ch19.generics;
2
3 public class GenericNode2<T, K> {
4     private T item1;
5     private K item2;
6
7     public T getItem1() {
8         return item1;
9     }
10
11     public void setItem1(T item1) {
12         this.item1 = item1;
13     }
14
15     public K getItem2() {
16         return item2;
17     }
18
19     public void setItem2(K item2) {
20         this.item2 = item2;
21     }
22 }
```

- Τα T και K είναι οι γενικοί τύποι και στη θέση τους μπορούν να περάσουν διάφοροι τύποι κλάσεων της Java ή κλάσεων ορισμένων από τον χρήστη



Δήλωση με δύο τύπους (2)

Προγραμματισμός με Java

```
1 package gr.aueb.cf.ch19.generics;
2
3 public class GenericNode2Demo {
4
5     public static void main(String[] args) {
6         GenericNode2<Integer, String> node = new GenericNode2<>();
7         node.setItem1(1);
8         node.setItem2("Alice");
9     }
10 }
```

- Η μόνη τεχνική διαφορά για τη δημιουργία ενός αντικειμένου με ένα τύπο σε σχέση με τη δημιουργία ενός αντικειμένου με δύο τύπους είναι η χρήση δύο τύπων μέσα στον τελεστή `<>`, π.χ. `<Integer, String>`



Ονόματα γενικών τύπων

Προγραμματισμός με Java

- Κατά σύμβαση οι παράμετροι τύπων δηλώνονται με ένα κεφαλαίο γράμμα, όπως παρακάτω.
 - E - Element (used extensively by the Java Collections Framework)
 - K - Key
 - N - Number
 - T - Type
 - V - Value
 - S,U,V etc. - 2nd, 3rd, 4th types



Γενικές μέθοδοι (1)

Προγραμματισμός με Java

- Μπορούμε επίσης –εκτός από κλάσεις και interfaces- να δηλώνουμε και γενικές μεθόδους. Ο παραμετρικός τύπος ορίζεται πριν τον επιστρεφόμενο τύπο

```
13  @      public static <T> void printArray(T[] array) {  
14          for (T item : array) {  
15              System.out.println(item);  
16          }  
17      }
```



Γενικές Μέθοδοι (2)

Προγραμματισμός με Java

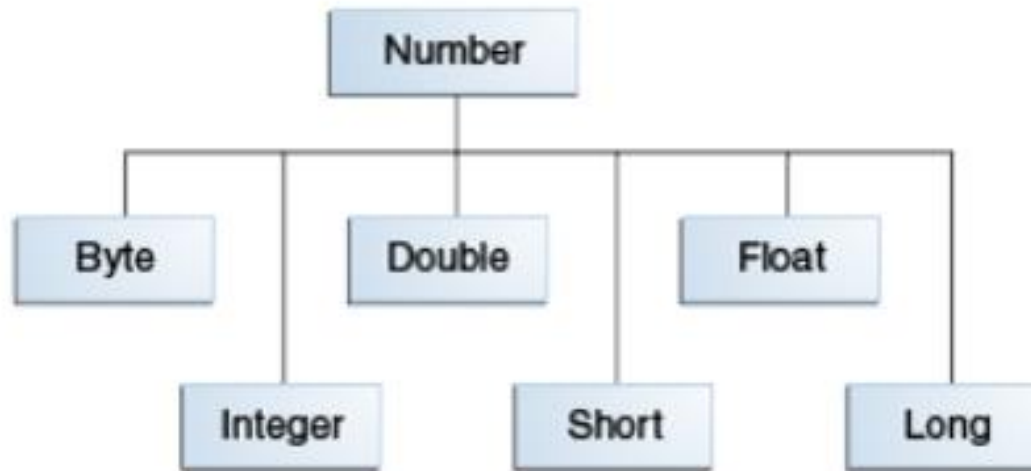
```
1 package gr.aueb.cf.ch19.generics;
2
3 public class GenericMethodClass {
4
5     public static void main(String[] args) {
6         Integer[] intArr = {1, 2, 3, 4, 5};
7         String[] strArr = {"Athens", "London", "Paris"};
8
9         printArray(intArr);
10        printArray(strArr);
11    }
12
13    @ public static <T> void printArray(T[] array) {
14        for (T item : array) {
15            System.out.println(item);
16        }
17    }
```

- Μπορούμε έτσι να εκτυπώνουμε διάφορους τύπους array



Ιεραρχία κλάσης Number

Προγραμματισμός με Java



- **Number** είναι μία κλάση της Java από την οποία κληρονομούν όλες οι wrapper κλάσεις που αναπαριστούν αριθμούς
- Υπάρχουν τέσσερις επιπλέον υποτύποι του **Number**
 - **BigDecimal** και **BigInteger** (για μεγάλη ακρίβεια)
 - **AtomicInteger** και **AtomicLong** (για multi-threaded)



Generic Wildcards

Προγραμματισμός με Java

- Τα wildcards στους παραμετρικούς τύπους μας επιτρέπουν μεγαλύτερο έλεγχο επί των τύπων παραμέτρων που δηλώνουμε
- Δύο κατηγορίες
 - Bounded (Φραγμένοι)
 - `<? extends type>`
 - `<? super type>`
 - Unbounded (Μη φραγμένοι)
 - `<?>`



Δηλώσεις πραγματικών τύπων

Προγραμματισμός με Java

- *NodeGeneric<Integer>*
- *NodeGeneric <String>*
- *NodeGeneric <?>*
- *NodeGeneric <? extends Number>*
- *NodeGeneric <? super Integer>*
- *NodeGeneric <Object>*

- Ο χαρακτήρας ? σημαίνει «Άγνωστος τύπος» (κατά τον χρόνο μεταγλώττισης)
- Ενώ το Object είναι υπερτύπος όλων των κλάσεων στην Java, το NodeGeneric<Object> δεν είναι υπερτύπος όλων παραμετρικών τύπων
- Με αυτή τη δήλωση NodeGeneric<Object> έχουμε τα προβλήματα που αναφέρθηκαν με *CastClassException*, ωστόσο μπορεί να χρησιμοποιηθεί αν χρειάζεται
- Υπερτύπος όλων των παραμετρικών τύπων είναι ο NodeGeneric<?>



Χαρακτήρας μπαλαντέρ ?

Προγραμματισμός με Java



```
19  @      public static void print(NodeGeneric<?> node) {  
20      |      System.out.println(node.getItem());  
21      |      }
```

- Στην *print()* δεν υπάρχουν εξαρτήσεις μέσα στο σώμα της (το ? εμφανίζεται στην επικεφαλίδα, αλλά όχι στο σώμα)



Άνω Φραγμένο ?

Προγραμματισμός με Java

```
19 @  public static void print(NodeGeneric<? extends Number> node) {  
20     System.out.println(node.getItem());  
21     }  
    
```

- Φραγμένος ? για να περιορίσουμε την εφαρμογή της print σε αριθμούς
- Το NodeGeneric μπορεί να περιέχει μόνο Number και υποκλάσεις



Κάτω φραγμένο ?

```
19 @      public static void print(NodeGeneric<? super Integer> node) {  
20         System.out.println(node.getItem());  
21     }
```

- Φραγμένος ? για να περιορίσουμε την εφαρμογή της print σε Integer, και υπερτύπους της κλάσης Integer



PECS

```
17  @ public static <T> void produceConsume(List<? super T> list, Iterable<? extends T> src) {  
18      for (T t : src) {  
19          list.add(t);  
20      }  
21  }
```

- Ένας μνημονικός κανόνας για το πότε χρησιμοποιούμε `extends` και πότε `super` είναι ο PECS (producer-extends, consumer-super)
- Producer είναι η οντότητα που παρέχει δεδομένα (input, όπως το `Iterable src`) και consumer η οντότητα που καταναλώνει δεδομένα (output, όπως η `List list`)



Άνω φράγματα σε τύπους

Προγραμματισμός με Java

```
1 package gr.aueb.elearn.generics.examples;
2
3 /**
4  * Ορίζει κλάση που περιέχει
5  * αντικείμενα Number.
6  *
7  * @param <T> τύπος/υποτύπος Number
8  */
9 public class NodeGenericBounded<T extends Number> {
10     private T item;
11
12     public T getItem() {
13         return item;
14     }
15
16     public void setItem(T item) {
17         this.item = item;
18     }
19 }
```

- Μπορούμε να **περιορίσουμε** και τις παραμέτρους τύπων
- Εδώ επιτρέπουμε μόνο **Number** και υποκλάσεις, δηλαδή αριθμούς

- Παραδείγματα δηλώσεων φραγμένων μεταβλητών τύπων:
 - **S extends T**
 - **T extends Number**
 - **E extends Number & Serializable**

Οι παράμετροι τύπων δεν μπορούν να έχουν κάτω φράγμα (super)



Άνω φράγματα σε μεθόδους

Προγραμματισμός με Java

```
1 package testbed.ch19;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.List;
6
7 public class Node<T> {
8     private T item;
9     private List<? extends T> items = new ArrayList<>();
10
11     public T getItem() { return item; }
12
13     public void setItem(T item) { this.item = item; }
14
15     @
16     public long getCount(List<? extends T> arrList) {
17         return arrList.size();
18     }
19
20     public List<? extends T> getItems() {
21         return Collections.unmodifiableList(items);
22     }
23
24     public void setItems(List<? extends T> items) {
25         this.items = new ArrayList<>(items);
26     }
27 }
28
29 }
```

- Το `? extends T` υποδηλώνει κληρονομικότητα
- Έχουμε ένα γενικό τύπο `T` και κλάσεις που κάνουν `extends T`
- Οπότε δημιουργούμε γενικές μεθόδους, όπως `getCount`, `setCount`



Παράδειγμα – Γενικές μέθοδοι

Προγραμματισμός με Java

```
6 public class Test {
7
8 public static void main(String[] args) {
9     var list = List.of(1, 2, 3, 4, 5);
10    var strList = Arrays.asList("Car ", "Cat ", "Camel");
11    Integer[] intArr = {1, 2, 3};
12
13    print(list);
14    System.out.println();
15    print(strList);
16    System.out.println();
17    print(intArr);
18 }
19
20 @ public static void print(List<?> list) {
21     list.forEach(System.out::print);
22 }
23
24 @ public static <T> void print(T[] arr) {
25     for (T arrayItem : arr) {
26         System.out.println(arrayItem);
27     }
28 }
29 }
```

- Η `List.of()` επιστρέφει unmodifiable list
- Η `Arrays.asList()` επιστρέφει structurally immutable list (όχι add, remove). Μόνο set επιτρέπεται για updates
- Η `print(List<?> list)` δεν έχει εξαρτήσεις στο σώμα της συνάρτησης και λειτουργεί ως πολυμορφική
- Η `print(T[] arr)` έχει εξαρτήσεις στο σώμα της συνάρτησης και είναι με γενικό τύπο



Παράδειγμα – Node<T>

Προγραμματισμός με Java

- Ορίζουμε ένα κόμβο που περιέχει μία τιμή και δύο δείκτες πριν και μετά

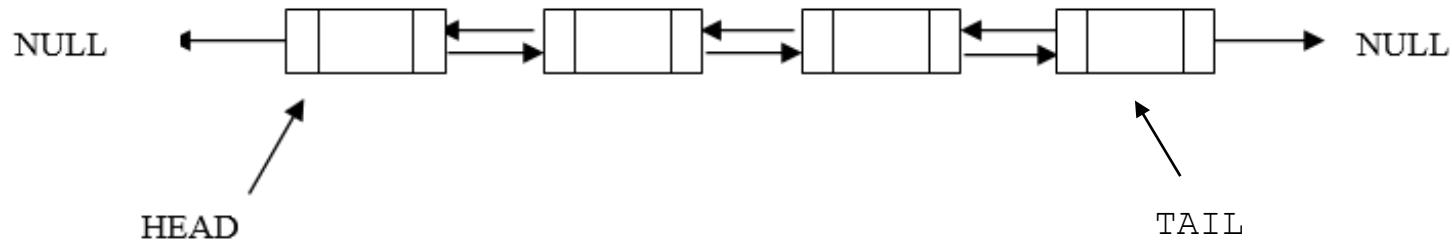
```
1 package gr.aueb.cf.generics;
2
3 /**
4  * The {@link Node} class implements a generic two-direction node.
5  * {@link #next} is a (self-)reference that points to the next node in a list
6  * {@link #prev} is a (self-)reference that points to the previous node in a list
7  * @param <T> is a generic type variable
8  */
9 public class Node<T> {
10     private T value;
11     private Node<T> next;
12     private Node<T> prev;
13
14     public Node() {}
15     public Node(T item) { this.value = item; }
16
17     public T getValue() { return value; }
18     public void setValue(T value) {...}
19     public Node<T> getNext() {...}
20     public void setNext(Node<T> next) {...}
21     public Node<T> getPrev() {...}
22     public void setPrev(Node<T> prev) {...}
23
24     @Override
25     public String toString() {...}
26 }
```



Διπλά συνδεδεμένη λίστα

Προγραμματισμός με Java

Διαγραμματικά, μία διπλά συνδεδεμένη λίστα μπορεί να παρασταθεί ως εξής:



Δυναμική Γραμμική Λίστα δύο διευθύνσεων

- Για να βελτιστοποιήσουμε τη διαδικασία της εισαγωγής και διαγραφής στο τέλος μπορούμε να προσθέσουμε και ένα δείκτη tail που να δείχνει στο τελευταίο στοιχείο



Doubly Linked List (1)

Προγραμματισμός με Java

```
1 package gr.aueb.cf.generics;
2
3 /**
4  * Doubly Linked List is a dynamic list implementation.
5  * It's composed of two private Node<T> instances and
6  * the required methods for insertion and deletion of
7  * nodes in the list
8  *
9  * @param <T> the generic type
10 */
11 public class DoublyLinkedList<T> {
12     private Node<T> head;
13     private Node<T> tail;
14
15     public DoublyLinkedList() {
16         head = tail = null;
17     }
18 }
```

- Μία λίστα αρχικοποιείται με `head = tail = null`



Doubly Linked List (2)

Προγραμματισμός με Java

```
19  /**
20   * Time complexity  $O(1)$ .
21   *
22   * @param t the item to be
23   */
24  public void insertFront(T t) {
25      Node<T> nodeToBeInserted = new Node<>();
26      nodeToBeInserted.setValue(t);
27      nodeToBeInserted.setPrev(null);
28      nodeToBeInserted.setNext(head);
29      head = nodeToBeInserted;
30      if (nodeToBeInserted.getNext() == null) {
31          tail = nodeToBeInserted;
32      } else {
33          nodeToBeInserted.getNext().setPrev(nodeToBeInserted);
34      }
35  }
```

- Η διαδικασία εισαγωγής στην αρχή σημαίνει ότι δημιουργούμε ένα κόμβο και κάνουμε τις κατάλληλες διευθετήσεις (prev -> null, next->head)
- Αν είναι ο μόνος κόμβος τότε πρέπει να δείχνει και το tail σε αυτόν αλλιώς αν δεν είναι ο μόνος θα πρέπει το prev του επόμενου (.next.prev) να δείχνει σε αυτόν



Doubly Linked List (3)

Προγραμματισμός με Java

```
37  /**
38   * Time complexity  $O(1)$  due to tail!
39   * If it was a one-direction list then
40   * insert at the end would be  $O(n)$ 
41   *
42   * @param t      the generic type
43   */
44  public void insertEnd(T t) {
45      Node<T> tmp = new Node<>();
46      tmp.setValue(t);
47      tmp.setPrev(tail);
48      tmp.setNext(null);
49      tail = tmp;
50      if (tmp.getPrev() == null) {
51          head = tmp;
52      } else {
53          tmp.getPrev().setNext(tmp);
54      }
55  }
```

- Για εισαγωγή το τέλος πάλι δημιουργούμε νέο κόμβο και ορίζουμε next-> null και prev-> tail
- Αν είναι ο μόνος κόμβος και το head πρέπει να δείχνει σε αυτόν
- Αλλιώς αν δεν είναι ο μόνος κόμβος, τότε θα πρέπει ο προηγούμενος να δείχνει σε αυτόν (tmp.prev.next = tmp)



Doubly Linked List (4)

Προγραμματισμός με Java

```
57      /**
58       * Time complexity O(1)
59       */
60      public void deleteFront() {
61          // If list contains only one node
62          if (head.getNext() == null) {
63              head = tail = null;
64          } else {
65              head = head.getNext();
66              head.setPrev(null);
67          }
68      }
```

- Η διαγραφή στην αρχή απλά θέτει το head στον επόμενο κόμβο και θέτει το prev του επόμενου κόμβου σε null αφού πλέον είναι ο πρώτος κόμβος
- Αν ωστόσο δεν υπάρχει επόμενος δηλαδή η λίστα περιέχει ένα μόνο κόμβο, τότε το head και το tail πρέπει να γίνουν null



Doubly Linked List (5)

Προγραμματισμός με Java

```
70  /**
71   * Time Complexity  $O(1)$  due to tail
72   * If it was a one-direction list then
73   * deleting the last node would be  $O(n)$ 
74   */
75  public void deleteEnd() {
76      // If list contains only one node
77      if (head.getNext() == null) {
78          head = tail = null;
79      } else {
80          tail = tail.getPrev();
81          tail.setNext(null);
82      }
83  }
```

- Η διαγραφή στο τέλος μετακινεί το tail στο prev node, και το next του prev node γίνεται null αφού είναι ο τελευταίος κόμβος
- Αν ωστόσο ο κόμβος είναι ο μόνο κόμβος της λίστας, τότε γίνεται head και tail = null



Doubly Linked List (6)

Προγραμματισμός με Java

```
82  /**
83   * Linear time complexity  $O(n)$ 
84   * Not very good when we have a lot of
85   * search work!
86   *
87   * @param t    the value to be searched
88   * @return    the node reference containing
89   *           the searched value or null
90   */
91  public Node<T> get(T t) {
92      for (Node<T> n = head; n != null; n = n.getNext()) {
93          if (n.getValue().equals(t)) {
94              return n;
95          }
96      }
97      return null;
98  }
```

- Η αναζήτηση στοιχείου γίνεται με συγκρίσεις, οπότε η πολυπλοκότητα χρόνου είναι στην χειρότερη περίπτωση $O(n)$



Doubly Linked List (7)

Προγραμματισμός με Java

```
103  /**
104   *  $O(n)$  time complexity
105   */
106  public void traverse() {
107      for (Node<T> n = head; n != null; n = n.getNext() ) {
108          System.out.println(n);
109      }
110  }
111
112  /**
113   *  $O(1)$  time complexity
114   *
115   * @return true if list is empty
116   */
117  public boolean isEmpty() {
118      return (head == null) && (tail == null);
119  }
120 }
```

- Η διάσχιση λίστας έχει πολυπλοκότητα χρόνου $O(n)$



Generics και Type Erasure

Προγραμματισμός με Java

```
1 package gr.aueb.elearn.chap13.generics;
2
3 public class Node<E> {
4     private E item;
5     private Node<E> next;
6
7     Node(E item) {
8         this.item = item;
9     }
10 }
```

Μεταγλωττιστής

```
12 // Type Erasure
13 class Node {
14     private Object item;
15     private Node next;
16
17     Node(Object item) {
18         this.item = item;
19     }
20 }
```

- Έστω η κλάση **Node<E>** (πάνω αριστερά) αναπαριστά ένα κόμβο που περιλαμβάνει ένα πεδίο **item** γενικού τύπου **E** καθώς ένα αυτό-αναφορικό πεδίο **next**
- Κατά το χρόνο μεταγλώττισης κάθε παραμετρικός τύπος αντικαθίσταται από το raw type (το όνομα του γενικού τύπου χωρίς ορίσματα, δηλ. **Node**) η μεταβλητή **E** αντικαθίσταται από τον τύπο **Object** (βλ. πάνω δεξιά). Αυτή η ιδιότητα ονομάζεται **Type Erasure**.
- Αφού λοιπόν πρώτα διενεργείται έλεγχος τύπων κατά το χρόνο μεταγλώττισης στη συνέχεια διενεργείται **Type Erasure**



Type Erasure σε γενικές κλάσεις με άνω φράγμα

Προγραμματισμός με Java

```
1 package gr.aueb.elearn.chap13.generics;
2
3 public class NumNode<E extends Number> {
4     private E item;
5     private NumNode<E> next;
6
7     NumNode(E item) {
8         this.item = item;
9     }
10 }
```

Μεταγλωττιστής

```
12 // Type Erasure
13 class NumNode {
14     private Number item;
15     private NumNode next;
16
17     NumNode(Number item) {
18         this.item = item;
19     }
20 }
```

- Η κλάση **NumNode<E extends Number>** αναπαριστά ένα κόμβο που περιλαμβάνει ένα πεδίο γενικού τύπου **E** που είναι υποκλάση του **Number** όπως αναφέρεται στην επικεφαλίδα της κλάσης με τη δήλωση **<E extends Number>** καθώς ένα αυτό-αναφορικό πεδίο **next**
- Κατά το type-erasure ο τύπος **E** διαγράφεται και κάθε μεταβλητή του τύπου **E** αντικαθίσταται κατά το χρόνο μεταγλώττισης με το **first bound** (δηλ. εδώ με την κλάση που κάνει **extends**) που εδώ είναι η κλάση **Number** και επομένως κατά το χρόνο εκτέλεσης παίρνουμε την εκδοχή που περιγράφεται δεξιά