ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Ο τύπος δεδομένων String

Αθ. Ανδρούτσος

Συμβολοσειρές

- Συμβολοσειρά (String) ή αλφαριθμητικό είναι μια ακολουθία χαρακτήρων
- Κυριολεκτικά Αλφαριθμητικά (literals) ορίζονται μέσα σε ***διπλά εισαγωγικά*** π.χ. "Athens"
- Οι συμβολοσειρές μπορούν να περιέχουν **οποιουσδήποτε εκτυπώσιμους χαρακτήρες** ή ειδικούς non-printing χαρακτήρες (με escape) ή UTF-8 χαρακτήρες (με \u)

Η κλάση String

- Ο τύπος δεδομένων **String** είναι ένας σύνθετος τύπος δεδομένων
- Για την ακρίβεια είναι μία **κλάση** της Java που περικλείει (wrap) ένα **πίνακα χαρακτήρων** (char[]) και παρέχεται από το package java.lang
- Όπως οι πίνακες έτσι και τα Strings ***ξεκινάνε από τη θέση 0 και φτάνουν μέχρι τη θέση length()-1***
- Παρατηρήστε ότι στα Strings το μήκος δίνεται από την **length()** με παρενθέσεις γιατί πρόκειται για μέθοδο ενώ στους πίνακες το length είναι ιδιότητα και όχι μέθοδος

Δημιουργία String (1)

- Όπως έχουμε αναφέρει, όλοι οι σύνθετοι τύποι, όπως οι πίνακες, **δημιουργούνται με new**
- Ωστόσο, υπάρχουν εξαιρέσεις. Στους πίνακες μπορούμε να δημιουργήσουμε πίνακα και με unsized initialization
- Εξαίρεση αποτελεί και ο τύπος String, όπου **μπορούμε να δημιουργήσουμε ένα String και με new αλλά και με απευθείας ανάθεση τιμής, π.χ. String s = "AUEB"**

Δημιουργία String (2)

- Ο πιο συνηθισμένος τρόπος δημιουργίας ενός String είναι η δημιουργία με ανάθεση τιμής, π.χ.
- ***String s = "alice";***
- Η δημιουργία String με new είναι σπάνια και χρησιμοποιείται μόνο σε ειδικές περιπτώσεις, όπως θα δούμε
- **String s = new String("alice");**

Δηλώσεις Strings

- Έστω για παράδειγμα η δήλωση ***String s = "alice";***
- Τότε είναι *s.length()* == 5 δηλαδή το String με περιεχόμενο "*alice*" έχει μήκος 5, και
- Ξεκινάει από τη θέση 0 και φτάνει μέχρι τη θέση 4, του πίνακα χαρακτήρων –char[] ο οποίος εσωτερικά υλοποιεί το String

Δήλωση και αρχικοποίηση


```
1   package testbed.ch7;
2
3   /**
4    * Two types of String declaration and
5    * initialization.
6    */
7   public class StringDeclaration {
8
9       public static void main(String[] args) {
10          String alice = "Alice";
11          String bob = new String("Bob");
12
13          System.out.println(alice + " and " + bob);
14          System.out.println("Total length = " + (alice.length() + bob.length()));
15      }
16  }
```

- Δηλώνουμε και αρχικοποιούμε στην γραμμή 10 με απευθείας εκχώρηση και στη γραμμή 11 με new
- Το μήκος του String δίνεται από την μέθοδο .length()

Char[] vs String

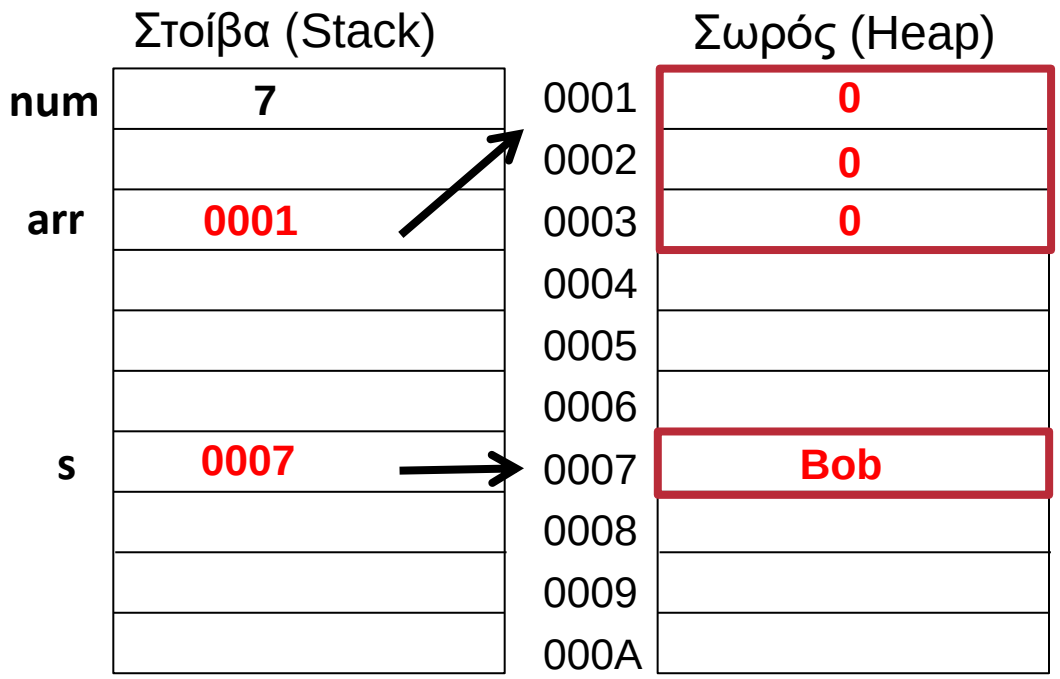

```java
package testbed.ch7;

public class StringVsCharArrayApp {

    public static void main(String[] args) {
        char[] alice = {'A', 'l', 'i', 'c', 'e'};
        String bob = "Bob";

        System.out.println(alice);
        System.out.println(bob);
    }
}
```

- Παρότι η κλάση String χρησιμοποιεί εσωτερικά ένα πίνακα χαρακτήρων, μας παρέχει ένα υψηλότερο μηχανισμό διαχείρισης αλφαριθμητικών, με χαρακτηριστικά καλύτερης και ευκολότερης διαχείρισης, παροχής μεθόδων, ασφάλειας, κλπ.

Stack vs Heap

- Να δούμε όμως πως ακριβώς ο μεταγλωττιστής της Java και το JVM διαχειρίζεται τις δηλώσεις Strings
- Όπως ήδη γνωρίζουμε η Java χρησιμοποιεί δύο λογικά μέρη στη μνήμη που διαχειρίζεται την αποθήκευση (storage) μεταβλητών, τη στοίβα (stack) και το σωρό (heap)
- Η **στοίβα** είναι μνήμη για **στατική** (σε χρόνο μεταγλώττισης) δέσμευση
- Ο **σωρός** είναι μνήμη για **δυναμική** (σε χρόνο εκτέλεσης) δέσμευση

Αναφορές / Δείκτες (1)



Έστω οι παρακάτω δηλώσεις και αρχικοποιήσεις:

- **int num = 7;**
  // Στατική δέσμευση num στη στοίβα
- **int[] arr = new int[3];**
  // Το arr στη στοίβα περιέχει τη διεύθυνση του 1ου στοιχείου του πίνακα που είναι στο σωρό
- **String s = "Bob";**
  // Το s στη στοίβα περιέχει τη διεύθυνση του αλφαριθμητικού που βρίσκεται στο σωρό

- Η αναφορική μεταβλητή arr, όπως έχουμε δει, είναι δείκτης (reference) προς τα περιεχόμενα του πίνακα. Κατά τον ίδιο τρόπο, η **αναφορική μεταβλητή s είναι δείκτης (reference) προς τα περιεχόμενα του String**
- Οι πρωταρχικοί τύποι (int, double, float, κλπ.) χρησιμοποιούν στατική δέσμευση. Οι σύνθετοι τύποι χρησιμοποιούν δυναμική δέσμευση

Αναφορές / Δείκτες (2)

- Θα πρέπει λοιπόν, στους σύνθετους τύπους δεδομένων, να διαχωρίσουμε: άλλο πράγμα οι δείκτες/αναφορές και άλλο οι τιμές στις οποίες δείχνουν
- Το **όνομα ενός String είναι δείκτης** στα περιεχόμενα του String, δηλαδή στο πραγματικό / κυριολεκτικό αλφαριθμητικό
- Αν δεν αρχικοποιήσουμε είτε με απευθείας ανάθεση ή με new ένα String s, τότε το s έχει τιμή **null** δηλαδή δεν έχει δεσμευτεί (allocate) χώρος στο σωρό
- String s; // s == null

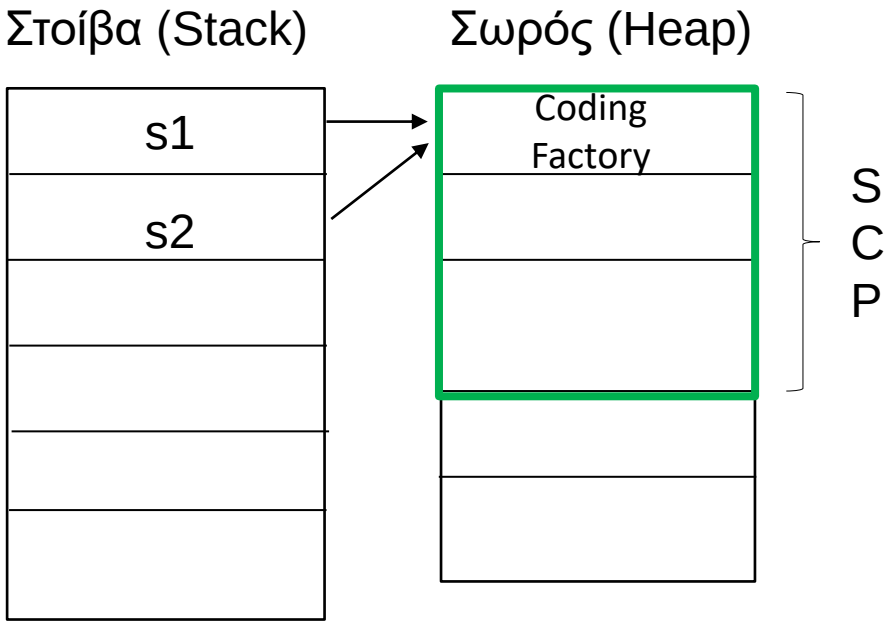
String Constant Pool (1)


```java
package testbed.ch7;

/**
 * Identical strings are stored only once, that
 * is, strings are interned in a special heap
 * area called String Constant Pool (SCP)
 */
public class StringConstantPool {

    public static void main(String[] args) {
        String s1 = "Coding Factory";
        String s2 = "Coding Factory";
    }
}
```

- Για λόγους efficiency **τα ίδια string literals** αποθηκεύονται **μία μόνο φορά** σε μια ειδική περιοχή της μνήμης, που αποτελεί μέρος του Heap, το **String Constant Pool** ή αλλιώς **Intern pool**

String Constant Pool (2)

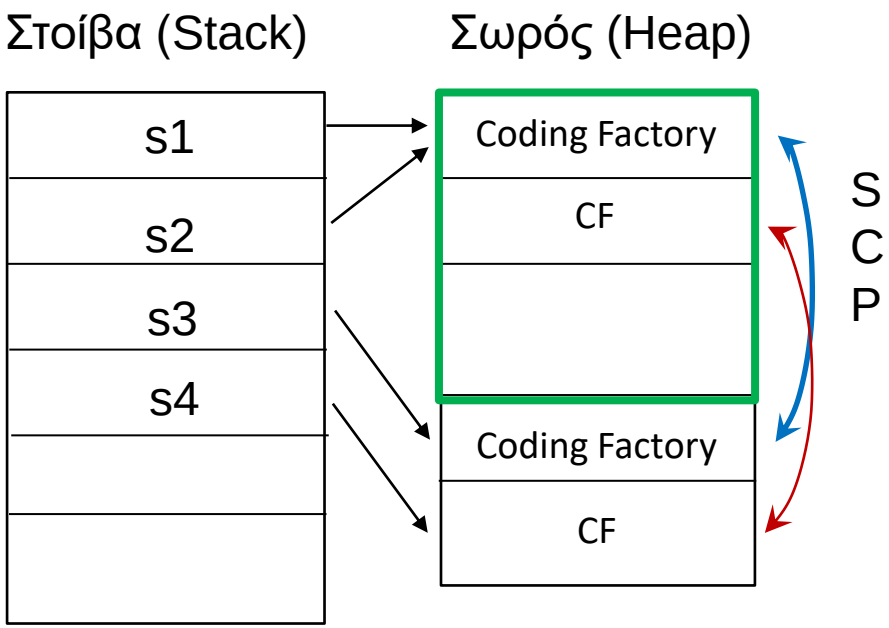

```
11        String s1 = "Coding Factory";
12        String s2 = "Coding Factory";
```



- Το "Coding Factory" αποθηκεύεται μία μόνο φορά (γίνεται interned) στο SCP (πράσινο πλαίσιο)
- Την 1η φορά στη γραμμή 11 γίνεται έμμεσα new, ενώ την επόμενη φορά στη γραμμή 12 δεν γίνεται new, αλλά το JVM ελέγχει το SCP και βλέπει ότι υπάρχει ήδη το "Coding Factory", οπότε και κάνει s2 = s1; δηλαδή αντιγραφή αναφορών

SCP και new

```
10    public static void main(String[] args) {
11        String s1 = "Coding Factory";
12        String s2 = "Coding Factory";
13        String s3 = new String("Coding Factory");
14        String s4 = new String("CF");
15    }
16 }
```



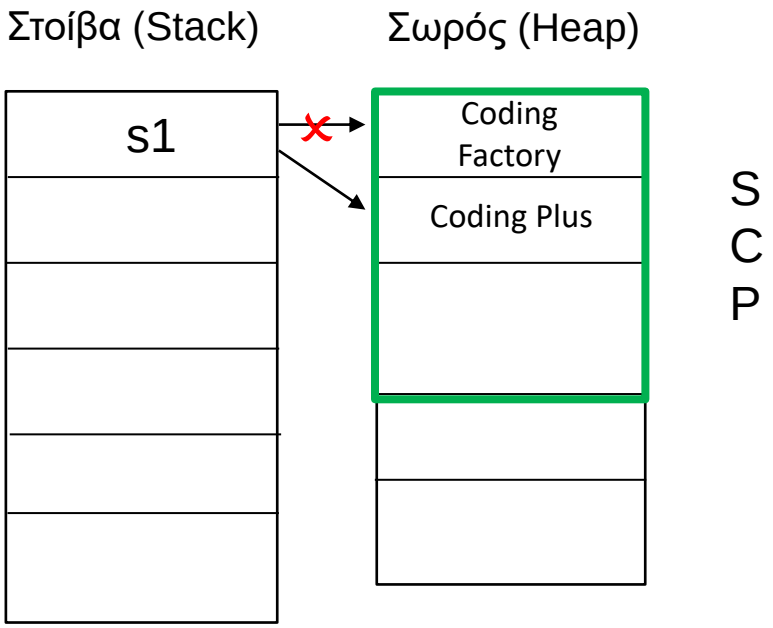
- Στη γραμμή 14 η new δημιουργεί νέο String στο Heap, "CF". Το string δεν υπάρχει στο SCP, αλλά δημιουργείται, άρα υπάρχουν δύο αντίγραφα και επομένως γίνεται μία περιττή πράξη new
- Στη γραμμή 13 η new δημιουργεί νέο String στο Heap "Coding Factory". Το string υπάρχει ήδη και στο SCP (αν δεν υπήρχε, όπως στην προηγούμενη περίπτωση, θα το δημιουργούσε)

Immutability

- Εφόσον η Java δίνει τη δυνατότητα τα strings να γίνονται cached (interned) με αντιγραφή αναφορών που δείχνουν στην ίδια θέση στο heap, δηλαδή δημιουργεί shallow copies, μήπως υπάρχει πρόβλημα όταν μία αναφορά αλλάξει τα περιεχόμενα ενός string, να υπάρχει ασυνέπεια στα περιεχόμενα που δείχνει η άλλη αναφορά;
- Όχι, γιατί τα stings στην Java είναι immutable. Οπότε, αν πάμε να αλλάξουμε ένα String, **απλά δημιουργούμε ένα άλλο καινούργιο**

Immutable (1)

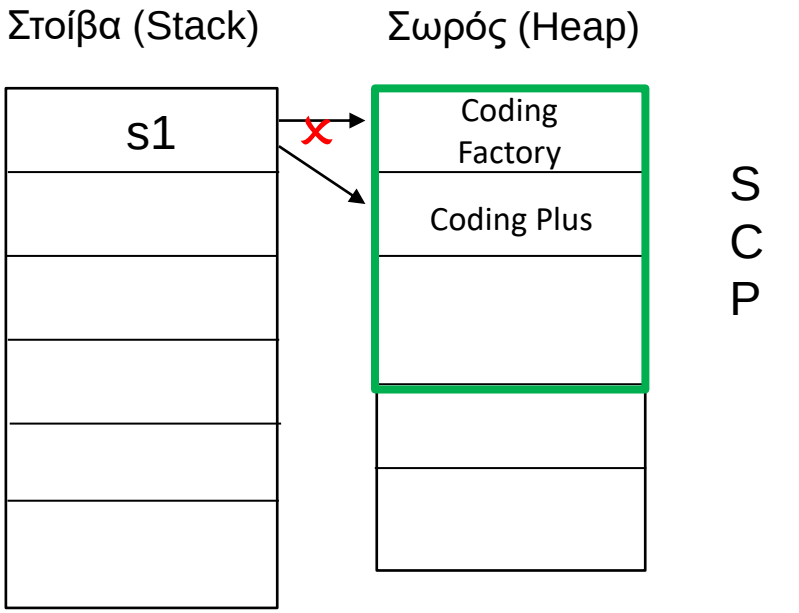

```java
package testbed.ch7;

/**
 * When a string gets a new value, it does
 * not change the contents of its memory storage.
 * Thus, strings can not be changes, they are
 * unchangeable or immutable.
 * Instead, a new string area is created in the heap
 * to store the new string.
 * The reference (s1) gets a new values (points to a
 * new position), while the old string remains unreferenced
 * and then is garbage collected.
 */
public class ImmutableApp1 {

    public static void main(String[] args) {
        String s1 = "Coding Factory";

        s1 = "Coding Plus";
    }
}
```



- Δίνοντας νέα τιμή περιεχομένου στο s1, δεν αλλάζει η παλιά τιμή αλλά αντ' αυτού δημιουργείται νέο string στο heap και το s1 δείχνει στο νέο string. Επομένως, τα strings είναι unchangeable ή αλλιώς immutable

Garbage Collector


```java
package testbed.ch7;

/**
 * When a string gets a new value, it does
 * not change the contents of its memory storage.
 * Thus, strings can not be changes, they are
 * unchangeable or immutable.
 * Instead, a new string area is created in the heap
 * to store the new string.
 * The reference (s1) gets a new values (points to a
 * new position), while the old string remains unreferenced
 * and then is garbage collected.
 */
public class ImmutableApp1 {

    public static void main(String[] args) {
        String s1 = "Coding Factory";

        s1 = "Coding Plus";
    }
}
```



- Η παλιά αναφορά στο "Coding Factory" παύει να υπάρχει και το string "Coding Factory" μένει χωρίς reference (ονομάζεται garbage, σκουπίδι)
- Μία ειδική διεργασία του JVM, ο Garbage Collector, διαγράφει τα "σκουπίδια" και τα αποδίδει (reclaims) πίσω στη μνήμη

Immutable (unchangeable)

- Όπως αναφέραμε, όταν στο s1 κάνουμε **νέα εκχώρηση δεν αλλάζει η τιμή του s1** (όπως ίσως θα περιμέναμε από την μέχρι τώρα εμπειρία μας στους πρωταρχικούς τύπους δεδομένων)
- Αντ' αυτού δημιουργείται νέος χώρος στο Heap
- Αυτό συμβαίνει γιατί ο τύπος String είναι **immutable**, **δεν μπορεί να γίνει modify** αλλά αντ' αυτού δημιουργείται νέο String

Immutability

- Το immutability, δηλαδή το να είναι τα Strings unchangeable είναι μία ιδιότητα σημαντική, μιας και αφού τα Strings χρησιμοποιούν ένα copy στο SCP, αν άλλαζαν και αν στην ίδια θέση έδειχναν περισσότεροι από ένας, θα υπήρχε ασυνέπεια, αν ένα reference μπορούσε να αλλάξει την τιμή (όπως στην περίπτωση των shallow copies)
- Ας φανταστούμε ότι σε δύο Strings s1 και s2 έχει εκχωρηθεί η τιμή "Coding Factory". Άρα (και επειδή στο SCP υπάρχει ένα μόνο "Coding Factory") και τα δύο s1 και s2 δείχνουν στην ίδια θέση μνήμης στο SCP
- Αν το s1 επιτρεπόταν να άλλαζε για παράδειγμα σε "Coding", τότε αυτό θα επηρέαζε και το String s2 το οποίο θα άλλαζε επίσης έμμεσα!

Immutable Κλάσεις

- Επομένως immutable είναι κάτι που δεν μπορεί να γίνει mutate (change). Χωρίς immutability, δεν μπορεί να υλοποιηθεί το caching (interning). Το immutability εγγυάται ότι τα cached literals δεν μπορούν να αλλάξουν και να οδηγήσουν σε ασυνέπειες
- Το immutability είναι μία σημαντική ιδιότητα μιας και προσδίδει προγραμματιστικά οφέλη. Ανάλογα με το use case είναι επιθυμητή ή όχι ιδιότητα
- Η Java Library παρέχει αρκετές immutable classes, όπως **String**, οι boxed primitive κλάσεις (**Integer, Byte, Short, Long, Float, Double, Boolean, Character**) καθώς και οι **BigInteger, BigDecimal**

SCP, Reuse, Performance

- Το γεγονός ότι τα strings είναι immutable επιτρέπει στην Java να χρησιμοποιεί μία μόνο θέση στο SCP (intern pool) και επομένως επιτρέπεται reuse χωρίς replication. Όπως αναφέραμε, το String interning είναι εφικτό λόγω του immutability
- Και τελικά λόγω αυτού (immutability) έχουμε τη δυνατότητα **reusing με καλύτερο performance σε όρους χώρου**

Ασφάλεια (1)

- Η κλάση String είναι μία από τις πιο πολυχρησιμοποιημένες κλάσεις στην Java και χρησιμοποιείται για να αποθηκεύσει ευαίσθητες πληροφορίες που προέρχονται από τους χρήστες, όπως username, password, string και εισάγονται στο πρόγραμμά (π.χ. ως παράμετροι σε SQL queries)

Ασφάλεια (2)

- Αν τα Strings ήταν mutable τότε θα υπήρχε ένα μεγάλο πρόβλημα ασφάλειας
- Θα μπορούσε κάποιος να τροποποιήσει ένα SQL Query που χρησιμοποιεί username/password ή να αλλάξει ένα όνομα αρχείου, και να δημιουργηθεί πρόβλημα ασφάλειας

Multithread και synchronization (1)

- Σε περιβάλλοντα multithread (δηλαδή πολλαπλών προγραμμάτων –threads- που μοιράζονται κοινές δομές μνήμης) δεν είναι επιθυμητό να επιτρέπονται *ταυτόχρονες (concurrent)* πράξεις που κάνουν mutate την κοινή δομή γιατί αν επιτρεπόταν κάτι τέτοιο τότε θα δημιουργούνται *race conditions*, δηλαδή μη-προβλέψιμες συμπεριφορές για το πιο thread θα έκανε πρώτο την αλλαγή
- Θα θέλαμε ιδανικά οι πράξεις mutate που γίνονται από τα threads σε κοινές δομές μνήμης να είναι atomic, δηλαδή αδιαίρετες, δηλαδή να μη μπορεί δύο threads ταυτόχρονα να αλλάζουν (mutate) μία θέση μνήμης

Multithread και synchronization (2)

- Ο γενικός μηχανισμός που επιτυγχάνει κάτι τέτοιο στις γλώσσες προγραμματισμού είναι το synchronization, δηλαδή να μην μπορούν οι πράξεις των threads να εκτελούνται concurrent αλλά ακολουθιακά, η μία μετά την άλλη (synchronized)
- Νοητικά μπορούμε να σκεφτόμαστε ότι όταν ένα thread εκτελεί την κρίσιμη περιοχή του (critical section) δηλαδή την περιοχή του κώδικα που κάνει mutate (insert, update, delete) θα πρέπει η κοινή περιοχή της μνήμης να κλειδώνει (lock) σαν δηλαδή να ανάβει ένα φανάρι κόκκινο όταν ένα thread εκτελεί το critical section και ξανά πράσινο μόλις τελειώσει

Multithread και synchronization (3)

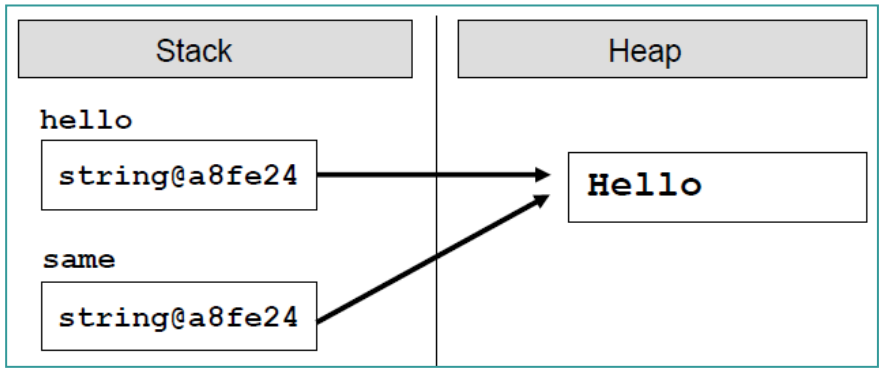
- Ωστόσο **όταν η κοινή δομή μνήμης είναι String** ο μηχανισμός synchronization δεν χρειάζεται γιατί τα Strings είναι immutable και δεν μπορούν οι mutate πράξεις να τα αλλάξουν
- Αν τα strings ήταν mutable τότε θα χρειαζόταν synchronization μιας και θα επιτρεπόταν πράξεις που θα έκαναν mutate
- Επομένως **τα strings λόγω του immutability είναι thread-safe**

Ασφάλεια Δεδομένων και Μεθόδων

- Επίσης και στο επίπεδο των μεθόδων θα θέλαμε να διασφαλίσουμε ότι δεν υπάρχουν side effects, όπως αλλαγή των τιμών των παραμέτρων εισόδου όταν πρόκειται για αναφορές, αλλαγή global data, αλλαγή static variables, input/output streams
- Θα θέλαμε δηλαδή για λόγους ασφάλειας οι μέθοδοί μας να είναι pure functions, δηλαδή απλά να επιστρέφουν μία τιμή ή να κάνουν αυτό πρέπει να κάνουν χωρίς side effects

Strings Interning

- Ανακεφαλαιώνοντας, η διαδικασία που περιγράψαμε με το SCP, ονομάζεται **Strings Interning**
- Αν σε κάποιο String δώσουμε τιμή που υπάρχει ήδη στο pool των Strings τότε ο νέος δείκτης θα δείξει στο ίδιο υπάρχον string




```
string hello = "Hello";
string same = "Hello";
```

String literal vs String object

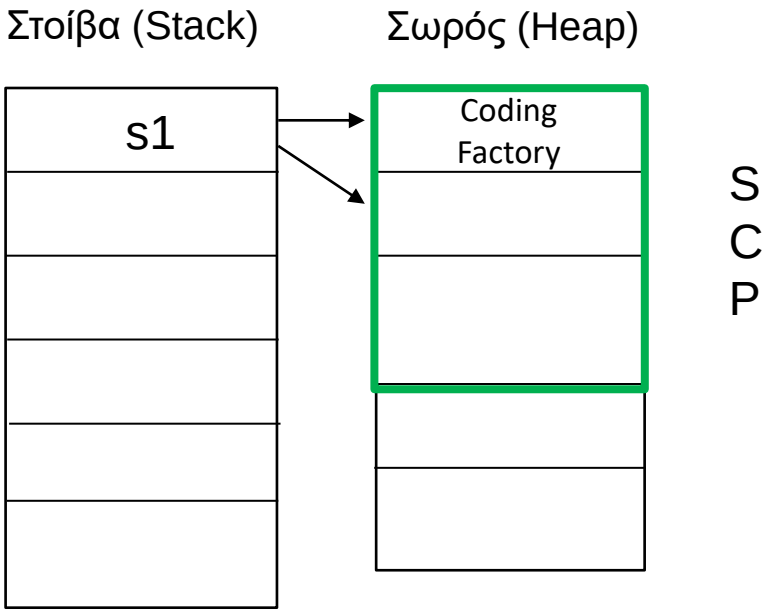
- Έστω η δήλωση String s1 = "Alice";
- Με τη δήλωση αυτή έμμεσα καλείται η μέθοδος **String.intern()** που κάνει το εξής: αν υπάρχει το string "Alice" στο SCP τότε το s1 (που είναι δείκτης) δείχνει στο υπάρχον "Alice" στο SCP, αλλιώς αν δεν υπάρχει, δημιουργείται
- Η δήλωση String s = new String("Bob"); δημιουργεί ένα αντικείμενο τύπου String στο heap αλλά αντιγράφει και στο SCP, **δηλαδή δημιουργεί δύο Strings**

Δήλωση String με new

- Δεδομένου ότι το new στα Strings αντιγράφει το String Literal στο SCP από το Heap δεν προσφέρει κάτι (απλά ξοδεύει περισσότερο χώρο γιατί υπάρχουν δύο αντίγραφα)
- Επομένως είναι προτιμότερο και προτείνεται να χρησιμοποιούμε δηλώσεις και αρχικοποιήσεις Strings, χωρίς την new, δηλαδή `String s = "Hello";`

Garbage Collection (Συλλογή Σκουπιδιών)


```
public class ImmutableApp1 {

    public static void main(String[] args) {
        String s1 = "Coding Factory";

        s1 = "Coding Plus";
    }
}
```



- Είδαμε ότι όταν δώσουμε νέα τιμή σε ένα string τότε το παλιό string στο heap μένει **χωρίς δείκτη και ονομάζεται σκουπίδι (garbage)**. Αν συμβούν πολλές τέτοιες περιπτώσεις, τότε η μνήμη heap γεμίζει με περιεχόμενα στα οποία δεν δείχνει κανένας δείκτης και απλά μειώνεται η χωρητικότητά της και η απόδοσή της
- Στην Java, οι προγραμματιστές δεν χρειάζεται να νοιάζονται για αυτές τις περιπτώσεις γιατί στο **JVM υπάρχει μία αυτόματη διεργασία που ονομάζεται Garbage collector** που καταστρέφει αυτά τα αντικείμενα που είναι δίχως δείκτες (απελευθερώνει τη μνήμη και την κάνει reclaim)

Πίνακες χαρακτήρων και Strings

- Όπως είπαμε η κλάση String αναπαριστά ένα **πίνακα χαρακτήρων** αλλά **δεν έχουμε άμεση πρόσβαση** στα στοιχεία αυτού του πίνακα, όπως είχαμε στα arrays
- Αν, για παράδειγμα, έχουμε το String s = "Hello" δεν μπορούμε να αναφερθούμε στα s[0] ή s[1] κλπ.
- Αντίθετα, χρησιμοποιούμε **μεθόδους της κλάσης String** για να έχουμε πρόσβαση στα στοιχεία ενός String
- Αυτές οι μέθοδοι χρησιμοποιούνται με το όνομα της μεταβλητής τύπου String και τον τελεστή τελεία (**dot notation**) – για παράδειγμα **s.charAt(0)** για τον 1ο χαρακτήρα του String

Input/Output (1)

- Έστω String str = "Bob". Εκτύπωση ενός String γίνεται με τους παρακάτω τρόπους:
  - System.out.print(str);
  - System.out.println(str);
  - System.out.printf("%s\n", str);
- Στην 1η περίπτωση εκτυπώνεται στην οθόνη το Bob
- Στην 2η περίπτωση εκτυπώνεται στην οθόνη το Bob και ο κέρσορας μετακινείται στην επόμενη γραμμή
- Στην 3η περίπτωση, όπως και στην 2η εκτυπώνεται στην οθόνη το Bob (το **%s** στην printf() σημαίνει String)

Input/Output (2)

- Για να διαβάσουμε ένα String από το πληκτρολόγιο χρησιμοποιούμε Scanner, όπως παρακάτω:
  - Scanner sc = new Scanner(System.in);
  - String s = sc.next();
  - String s = sc.nextLine();
- Η **.next()** διαβάζει χαρακτήρες μέχρι να βρει white space (κενό χαρακτήρα / tab ή αλλαγή γραμμής)
- Η **.nextLine()** διαβάζει χαρακτήρες μέχρι να βρει μόνο τέλος γραμμής. Άρα διαβάζει και τα κενά και τα tab

Παράδειγμα I/O από πληκτρολόγιο


```java
package testbed.ch7;

import java.util.Scanner;

/**
 * Reads strings from the standard input.
 */
public class StringInputApp {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s;

        System.out.println("Please insert a string that ends with whitespace (enter/tab/space)");
        s = sc.next();
        sc.nextLine();          // consumes the new line character(s)
        System.out.println(s);

        System.out.println("Please insert a new string that ends with enter");
        s = sc.nextLine();
        System.out.println(s);
    }
}
```

- Διαβάζουμε με Scanner ένα String από το stdin και εκτυπώνουμε
- Την 1η φορά διαβάζουμε με .next() ένα string που δεν έχει κενά ή tabs
- Την 2η φορά διαβάζουμε όλους τους χαρακτήρες μέχρι να βρούμε αλλαγή γραμμής

Πράξεις σε Strings

- Έχουμε δει πως κάνουμε populate με απ’ ευθείας ανάθεση (και με new αλλά αυτό είναι κάτι που δεν έχει ουσιαστικό νόημα όπως είπαμε)
- Θα δούμε πως κάνουμε read έναν-ένα τους χαρακτήρες ενός string με charAt() ή αναζητούμε ένα χαρακτήρα ή substring μέσα στο string
- Update και delete δεν μπορούμε να κάνουμε άμεσα καθότι τα strings είναι immutable. Ωστόσο μπορούμε έμμεσα να κάνουμε κάτι παρόμοιο, δημιουργώντας νέα strings

Άλλες βοηθητικές πράξεις

- Άλλες βοηθητικές (utility) πράξεις
  - Σύγκριση ισότητας (==, equals)
  - Σύγκριση ανισότητας, μικρότερου, μεγαλύτερου (compareTo)
  - Αντιγραφή String
  - Συνένωση Strings (+, concat)
  - Parsing του String σε substrings, που διαχωρίζονται με κενά ή άλλους ειδικούς χαρακτήρες
  - Αφαίρεση πιθανών κενών στην αρχή και το τέλος (trim)
  - Μετατροπές, σε πεζά (toLowerCase), σε κεφαλαία (toUpperCase), σε αριθμούς, π.χ. σε int από String με **Integer.ParseInt("a string")**, σε Strings από άλλους τύπους, με **String.valueOf()**
- https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Populate


```java
package testbed.ch7;

/**
 * Populate a string.
 */
public class PopulateApp {

    public static void main(String[] args) {
        String username = "thanos";
    }
}
```

- Δήλωση και αρχικοποίηση (populate) ενός string

Διάβασμα και διάσχιση (1)

- Μπορούμε να χρησιμοποιήσουμε μία for και με charAt() να διαβάσουμε τα περιεχόμενα ενός String
- Η **for-each (enhanced for) δεν δουλεύει για strings** αλλά μπορούμε να μετατρέψουμε σε char array, **str.toCharArray()**

For-each με char array


```java
package testbed.ch7;

/**
 * Prints a string as char array
 * with for-each.
 */
public class ForEachApp {

    public static void main(String[] args) {
        String s = "Coding Factory";

        for (char ch : s.toCharArray()) {
            System.out.print(ch + " ");
        }

        System.out.println("\u2764");
    }
}
```

```
Run:  ForEachApp
"C:\Program Files\Amazon Corretto\jdk11.0.10_
C o d i n g   F a c t o r y ❤

Process finished with exit code 0
```

- Η for-each δεν δουλεύει με strings
- Μπορούμε ωστόσο να μετατρέψουμε το string σε char-array και να κάνουμε traverse με for-each

Δημιουργία και Διάσχιση(2)


```java
package testbed.ch7;

/**
 * Traverses, reads and prints a string,
 * character by character.
 */
public class StringAccessApp {

    public static void main(String[] args) {
        String s = "Coding Factory";

        for (int i = 0; i < s.length(); i++) {
            System.out.print(s.charAt(i) + " ");
        }

        System.out.println();

        // reverse order
        for (int i = s.length() - 1; i >= 0; i--) {
            System.out.print(s.charAt(i) + " ");
        }
    }
}
```

- Η charAt(index) επιστρέφει τον χαρακτήρα ενός String που βρίσκεται στη θέση index του String
- Η 1η for εκτυπώνει ένα-ένα όλους τους χαρακτήρες του String
- Η 2η for εκτυπώνει σε αντίστροφη σειρά

Αναζήτηση χαρακτήρων / substrings

- Έστω String s = "Alice";
- Η κλάση String παρέχει την μέθοδο **s.contains(str)** που ελέγχει **αν ένα String str** περιέχεται μέσα στο s και επιστρέφει boolean
- indexOf(int character), indexOf(String substring)
- indexOf(int character, int fromIndex), indexOf(String substring, int fromIndex)
  - Επιστρέφουν τη θέση που βρίσκεται ένας χαρακτήρας ή ένα substring μέσα σε ένα String

Αναζήτηση χαρακτήρα


```
5   public static void main(String[] args) {
6       String s = "Athens University of Economics and Business";
7       int index = -1;
8
9       if (s.contains("Economics")) System.out.println("Bingo");
10
11      index = s.indexOf("Uni");
12      System.out.println("Start Index of Uni: " + index);
13
14      index = s.indexOf('t');
15      System.out.println("Start Index of 't': " + index);
16
17      index = s.lastIndexOf('t');
18      System.out.println("Start Index of 't': " + index);
19
20      index = s.indexOf('s');
21      System.out.println("Start Index of 's': " + index);
22
23      index = s.indexOf('s', 7);
24      System.out.println("Start Index of 's': " + index);
25
26      if (s.startsWith("Athens")) System.out.println("Bongo");
27      if (s.startsWith("Uni", 7)) System.out.println("Bingo");
28  }
29  }
```

- Με την contains() ελέγχουμε αν υπάρχει το "Economics" και εκτυπώνουμε κατάλληλο μήνυμα
- Η indexOf() επιστρέφει τη θέση μέσα στο String που περιέχει τον χαρακτήρα ή το substring
- Μπορεί να λάβει και 2η παράμετρο που είναι το start index, από που δηλαδή να ξεκινήσει να ψάχνει
- Η lastIndex() επιστρέφει την τελευταία θέση από όπου ξεκινάει το substring, αν υπάρχουν πολλαπλές εμφανίσεις
- Η startsWith() επιστρέφει boolean αν το string ξεκινάει με το substring. Η 2η παράμετρος, αν υπάρχει, είναι το start index

Εξαγωγή substring - parsing

- s.substring(int beginIndex) – Επιστρέφει ένα substring από τη θέση beginIndex μέχρι το τέλος του string
- s.substring(int beginIndex, int endIndex) - Επιστρέφει από beginIndex έως endIndex - 1
  - Αν είναι s.substring(1, 4) επιστρέφει από τη θέση 1 μέχρι τη θέση 3
  - Αν είναι s.substring(2) επιστρέφει από 2 μέχρι το τέλος (s.length() - 1)

Εκτύπωση με substring


```
1  package gr.aueb.than.ch7;
2
3  /**
4   * Εκτυπώνει ένα String σε μορφή substrings
5   * ενός χαρακτήρα τη φορά.
6   *
7   * @author A. Androutsos
8   */
9  public class SubstringDemo {
10
11     public static void main(String[] args) {
12         String s = "This is AUEB";
13
14         for (int i = 0; i < s.length(); i++) {
15             System.out.print(s.substring(i, i+1));
16         }
17     }
18 }
```

- Μέσα σε μια for εκτυπώνουμε κάθε χαρακτήρα του string με την substring(i, i+1) η οποία εκτυπώνει χαρακτήρα-χαρακτήρα

Αντικατάσταση και Διαγραφή


```java
package testbed.ch7;

public class ReplaceDeleteApp {

    public static void main(String[] args) {
        String firstname = "Maria-Helen";
        String simpleName;
        String nickname;
        String oneName;

        // Replaces dash with whitespace
        simpleName = firstname.replace("-", " ");

        // Replaces with empty string (string with length() == 0)
        // Thus essentially deletes
        nickname = firstname.replace("-", "");
        System.out.println(nickname);

        // Replaces with empty string - Deletes
        oneName = firstname.replace("-Helen", "");
        System.out.println(oneName);
    }
}
```

- Η replace αντικαθιστά ένα χαρακτήρα με ένα άλλο ή ένα substring με ένα άλλο
- Αν αντικαταστήσουμε με το κενό string, τότε διαγράφουμε
- Επίσης, η replaceAll() αντικαθιστά regex με ένα replacement string

Immutable - Αμετάβλητα

- Η replace δεν αλλάζει τα περιεχόμενα ενός String, αλλά δημιουργεί νέο String με τα καινούργια περιεχόμενα
- Ο λόγος είναι –όπως έχουμε αναφέρει- πως τα Strings είναι ***immutable δηλαδή αμετάβλητα***
- Δεν μπορούμε για παράδειγμα να αλλάξουμε την τιμή ενός χαρακτήρα του String. `s[1] = 'a';` Δίνει: **Λάθος μεταγλώττισης**
- Αν θέλουμε να αλλάξουμε την τιμή ενός String τότε **απλώς δημιουργούμε ένα νέο String**

Συνένωση


```java
package testbed.ch7;

public class ConcatApp {

    public static void main(String[] args) {
        String firstname = "Athanassios";
        String lastname = "Androutsos";
        String title = "Dr.";
        String fullName1, fullName2, fullName3, fullName4;

        // + operator is overloaded in Java.
        // When applied to strings is considered concat
        fullName1 = firstname + lastname;
        fullName2 = title + firstname + lastname;

        fullName3 = firstname.concat(lastname);
        fullName4 = title.concat(firstname).concat(lastname);       // method chain

        System.out.printf("Firstname: %s, Lastname: %s", firstname, lastname);
        System.out.printf("Full name 1: %s", fullName1);
        System.out.printf("Full name 2: %s", fullName2);
        System.out.printf("Full name 3: %s", fullName3);
        System.out.printf("Full name 4: %s", fullName4);
    }
}
```

- Ο τελεστής + συνενώνει δύο ή περισσότερα Strings
- Το ίδιο κάνει και η μέθοδος String.concat()

Ισότητα strings (1)

- String1 **==** (ή !=) String2 – Δεν συγκρίνουν το περιεχόμενο των Strings αλλά τους δείκτες σε Strings. Μπορεί s1 == s2 να είναι false αλλά τα s1, s2 να δείχνουν σε ίδιο περιεχόμενο
- Αν θέλουμε να συγκρίνουμε περιεχόμενα, συγκρίνουμε με την equals. String1.equals(String2) – ***Συγκρίνει το περιεχόμενο δύο Strings, επιστρέφει true αν είναι ίσα, false αν είναι άνισα***
- *Αν θέλουμε η σύγκριση να αγνοεί πεζά-κεφαλαία μπορούμε να χρησιμοποιούμε την* ***equalsIgnoreCase()***

Ισότητα Strings (2)


```java
package testbed.ch7;

/**
 * String.equals() just checks for equality
 * or inequality. Not less or greater than.
 */
public class EqualsApp {

    public static void main(String[] args) {
        String s1 = "Athens";
        String s2 = "Athens";
        String s3 = "London";
        String s4 = "ATHENS";

        boolean isEqual;

        isEqual = s1.equals(s2);            // isEqual is true
        isEqual = s1.equals(s3);            // isEqual is false
        isEqual = s1.equalsIgnoreCase(s4);  // isEqual is true
    }
}
```

- Επομένως, προσοχή!
- Σύγκριση Strings κάνουμε με την **equals**, όχι την ==

Σύγκριση

- Η equals επιστρέφει boolean και ελέγχει μόνο ισότητα/ανισότητα. Αν θέλουμε να ελέγχουμε την σχέση μικρότερο, ίσο, μεγαλύτερο, τότε μπορούμε να χρησιμοποιούμε την compareTo
  - **String1.compareTo(String2)** -- επιστρέφει 0 αν String1.equals(String2), <0 αν String1<String2 και >0 αν String1>String2
  - **String1.compareToIgnoreCase(String2)** – όπως παραπάνω αλλά θεωρεί ίδια πεζά-κεφαλαία γράμματα (άρα Alice θεωρείται ίδιο με alice)

Βάση συγκρίσεων

- Οι συγκρίσεις με την compareTo και γενικότερα μεταξύ Strings γίνονται λεξικογραφικά με βάση τον κώδικα ASCII
- Επομένως το "athens" είναι μεγαλύτερο από το "Athens" καθώς το ordinal number του a είναι το 97 ενώ του A το 65

Σύγκριση με compareTo


```
1   package testbed.ch7;
2
3   /**
4    * String.compareTo() and
5    * String.compareToIgnoreCase()
6    */
7   public class CompareApp {
8
9       public static void main(String[] args) {
10          String s1 = "Athens";
11          String s2 ="athens";
12
13          if (s1.compareTo(s2) < 0) System.out.println("'a'thens is greater than 'A'thens");
14          if (s1.compareToIgnoreCase(s2) == 0) System.out.println("Are equal ignoring case");
15      }
16  }
```

- Επειδή το 'a' είναι μεγαλύτερο από το 'A' με βάση το σύστημα ASCII, το athens είναι μεγαλύτερο από το Athens

Αντιγραφή


```java
package testbed.ch7;

public class CopyStringApp {

    public static void main(String[] args) {
        String s = "Coding Factory";
        String clone;

        clone = s;
    }
}
```

- Εφόσον τα strings είναι immutable η αντιγραφή δεικτών είναι συνεπής αντιγραφή. Δεν υπάρχει το πρόβλημα που είδαμε στα shallow copies των πινάκων
- Αν δηλαδή αλλάξουμε το clone, δεν αλλάζει το s, μιας και δημιουργείται νέο String

Typecasts (1)

- Από άλλους τύπους σε String
  - Για σύνθετους τύπους. ***toString()*** – Μετατρέπει αντικείμενα κλάσεων που δεν είναι String, σε String (θα την δούμε αυτή τη μέθοδο στο πλαίσιο της κληρονομικότητας)
  - Για πρωταρχικούς τύπους. Χρησιμοποιούμε την ***String.valueOf()***

Typecasts (2)

- Από Strings σε αριθμούς, όπως για παράδειγμα int
  - Η ***Integer.parseInt(str)*** μετατρέπει ένα string που αναπαριστά ακέραιο, σε ακέραιο τύπου int, π.χ. Integer.parseInt("15") όπου το "15" μετατρέπετε σε 15. Αντίστοιχα έχουμε Long.parseLong(), Float.parseFloat(), Double.parseDouble(), κλπ
  - Τυχόν προσπάθεια μετατροπής σε ακέραιο μη αριθμητικών χαρακτήρων δημιουργεί εξαίρεση NumberFormatException, π.χ. Integer.parseInt("ABC")

Typecast (3)


```java
package testbed.ch7;

import java.util.Scanner;

/**
 * Typecast from String to int. If the string
 * is not a valid int then NumberFormatException
 * is thrown.
 */
public class TypecastsApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String lexeme;
        int num;

        System.out.println("Please insert an int:");
        lexeme = in.next();
        num = Integer.parseInt(lexeme);

        System.out.println("Num is: " + num);
    }
}
```

- Από το standard input και από τα αρχεία διαβάζουμε πάντα strings
- Η nextInt() του scanner μετατρέπει αυτόματα τα strings σε ints
- Αν όμως διαβάσουμε κανονικά με .next() τότε για να μετατρέψουμε σε int χρησιμοποιούμε την Integer.parseInt()

Convert σε String


```java
package testbed.ch7;

/**
 * Converts primitives to String
 */
public class PrimitiveToString {

    public static void main(String[] args) {
        int num = 5;
        float f = 3.15F;
        String s1, s2;

        s1 = String.valueOf(num);
        s2 = String.valueOf(f);

        System.out.println(s1);
        System.out.println(s2);
    }
}
```

- Με την .valueOf() που είναι static μέθοδος της κλάσης String μετατρέπουμε από άλλους τύπους σε String

Μετατροπές (1)

- s.toLowerCase – Μετατρέπει σε πεζά γράμματα
- s.toUpperCase – Μετατρέπει σε κεφαλαία γράμματα
- s.trim() – Αφαιρεί κενά διαστήματα πριν και μετά το αλφαριθμητικό

Μετατροπές (2)


```java
package testbed.ch7;

/**
 * Convert to lowercase, uppercase, trimmed.
 */
public class LowerUpperTrimApp {

    public static void main(String[] args) {
        String s = "  Athens University of Economics and Business   ";
        String lowercase;
        String upperCase;
        String trimmed;

        lowercase = s.toLowerCase();
        upperCase = s.toLowerCase();
        trimmed = s.trim();

        System.out.printf("Lexeme: %s, Lowercase: %s\n", s, lowercase);
        System.out.printf("Lexeme: %s, Uppercase: %s\n", s, upperCase);
        System.out.printf("Lexeme: %s, Trimmed: %s\n", s, trimmed);
    }
}
```

- Η διαδικασία μετατροπής σε πεζά ή κεφαλαία είναι διαδικασία κανονικοποίησης (normalization)
- Θα μπορούσε να λειτουργήσει εναλλακτικά της equalsIgnoreCase(), να μετατρέπουμε δηλαδή δύο strings που θέλουμε να συγκρίνουμε με non-sensitive τρόπο, σε κεφαλαία και να συγκρίνουμε
- Η trim() αποκόπτει τα whitespaces στην αρχή και το τέλος του String

.repeat JDK11


```java
package testbed.ch7;

/**
 * Prints repeated stars.
 */
public class RepeatMethod {

    public static void main(String[] args) {

        // 20 horizontal stars
        System.out.println("*".repeat(20));

        // 10x10 stars
        for (int i = 1; i <= 10; i++) {
            System.out.println("*".repeat(10));
        }
    }
}
```

- Στην Java11 παρέχεται από την κλάση String και η μέθοδος .repeat(count) που επαναλαμβάνει το string, count φορές

Άλλες πράξεις σε Strings


```java
package testbed.ch7;

/**
 * Demonstrates string length.
 */
public class LengthIsEmptyApp {

    public static void main(String[] args) {
        String s1 = "Coding Factory";
        String s2 = "";
        boolean isEmpty;

        System.out.printf("String: %s, Length: %d\n", s1, s1.length());

        // Empty -> length == 0
        isEmpty = s2.isEmpty();
        System.out.printf("s2 is empty: %s, s2 length is: %d", isEmpty,  s2.length());
    }
}
```

- s.length() -- το μήκος του String
- s.isEmpty () -- boolean, ελέγχει αν το String έχει μηδενικό μήκος

String[] split(String regex)


```java
package gr.aueb.than.ch7;

public class StringSplit {

    public static void main(String[] args) {
        String s = "Athens University of Economics and Business";

        String[] tokens = s.split(" ");

        for (String token : tokens) {
            System.out.println(token);
        }
    }
}
```

```
Run: StringSplit
"C:\Program Files\Java\jdk1.8.0_40\bin\java.exe" ...
Athens
University
of
Economics
and
Business

Process finished with exit code 0
```

- Διασπά το String σε επιμέρους substrings (tokens) με delimiter όποιο delimiter ορίσουμε (στο παράδειγμα είναι το κενό) και επιστρέφει ένα πίνακα των επιμέρους tokens

String.format() Μορφοποίηση


```java
package gr.aueb.cf.ch7;

/**
 * Formats primitives to string.
 */
public class StringFormatApp {

    public static void main(String[] args) {
        char row = 'B';
        int col = 8;

        String seat = String.format("%c%02d", row, col);

        System.out.println(seat);
    }
}
```

- Η String.format() λειτουργεί όπως η printf
- Όμως δεν εκτυπώνει αλλά επιστρέφει ένα String
- Το %02d σημαίνει μορφοποίηση του int σε διάστημα δύο ψηφίων όπου αν υπάρχει μόνο ένα ψηφίο το 1ο διάστημα να είναι 0 και όχι κενό διάστημα, όπως θα ήταν αν βάζαμε απλά %2d

StringBuilder (1)

- Έτοιμη κλάση της Java που χρησιμοποιούμε όταν έχουμε πολλές πράξεις συνένωσης
- Η **συνένωση strings είναι πολύ χρονοβόρα** διαδικασία δεδομένου ότι τα Strings είναι immutable και σε κάθε συνένωση δημιουργείται νέο string
- Η κλάση StringBuilder είναι mutable

StringBuilder (2)

- Ο StringBuilder είναι σαν πίνακας από chars, που όμως είναι mutable και μεγαλώνει το μέγεθός του αυτόματα όταν προσθέτουμε νέους χαρακτήρες
- Για να προσθέσουμε νέους χαρακτήρες στο τέλος της ουράς του StringBuilder, χρησιμοποιούμε την append(i) που προσθέτει ένα στοιχείο i στο τέλος του StringBuilder

String vs StringBuilder (1)


```java
package gr.aueb.cf.ch7;

/**
 * Concatenates strings efficiently
 * with {@link StringBuilder}.
 */
public class StringBuilderApp {

    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        String concatStr = "";
        long timeStart = 0L;
        long timeEnd = 0L;
        double stringElapsedTime = 0.0;
        double sbElapsedTime = 0.0;

        timeStart = System.currentTimeMillis();
        for (int i = 1; i <= 15000; i++) {
            concatStr = concatStr + i;
        }
        timeEnd = System.currentTimeMillis();
        stringElapsedTime =  (timeEnd - timeStart) / 1000.0;
```

- Μετράμε το χρόνο συνένωσης με String
- Χρησιμοποιούμε την *currentTimeMillis()* που μετράει τον χρόνο σε milliseconds από το UNIX Epoch (1/1/1970 00:00:00)
- Μετατρέπουμε σε seconds διαιρώντας με το 1000.0 (βάζουμε .0 ώστε να μετατραπεί σε double) και η διαίρεση να επιστρέψει double, αλλιώς θα επέστρεφε int

String vs StringBuilder (2)


```
23
24          timeStart = System.nanoTime();
25          for (int i = 1; i <= 15000; i++) {
26              sb.append(i);
27          }
28          timeEnd = System.nanoTime();
29          sbElapsedTime =  (timeEnd - timeStart) / 1_000_000_000.0;
30
31          System.out.println("String concat time: " + stringElapsedTime + " seconds");
32          System.out.println("String Builder concat time: " + sbElapsedTime + " seconds");
33      }
34  }
```

- Μετράμε το χρόνο συνένωσης με StringBuilder. Για λόγους επίδειξης χρησιμοποιούμε *nanoTime()*. Είναι παρόμοια με την *currentTimeMillis()* , μετράει nanoseconds από το Epoch
- Μετατρέπουμε σε seconds διαιρώντας με το 1,000,000,000.0 (βάζουμε .0 ώστε να μετατραπεί σε double) και η διαίρεση να επιστρέψει double, αλλιώς θα επέστρεφε int

Παράμετρος args της main() Παράμετροι γραμμής εντολών

- Ο συμβολισμός String[] args στην main() ορίζει ένα πίνακα συμβολοσειρών που αποθηκεύει τις παραμέτρους της γραμμής εντολών
- Αν εκτελέσουμε ένα πρόγραμμα Java στην γραμμή εντολών (Command Line Interface - CLI) και περάσουμε παραμέτρους, τότε οι παράμετροι αποθηκεύονται αυτόματα στον πίνακα args
- Για παράδειγμα αν ένα πρόγραμμα hello.java το μεταγλωττίσουμε (με javac hello.java) και εκτελέσουμε (με java hello CF) τότε το CF θα είναι η $1^{η}$ παράμετρος

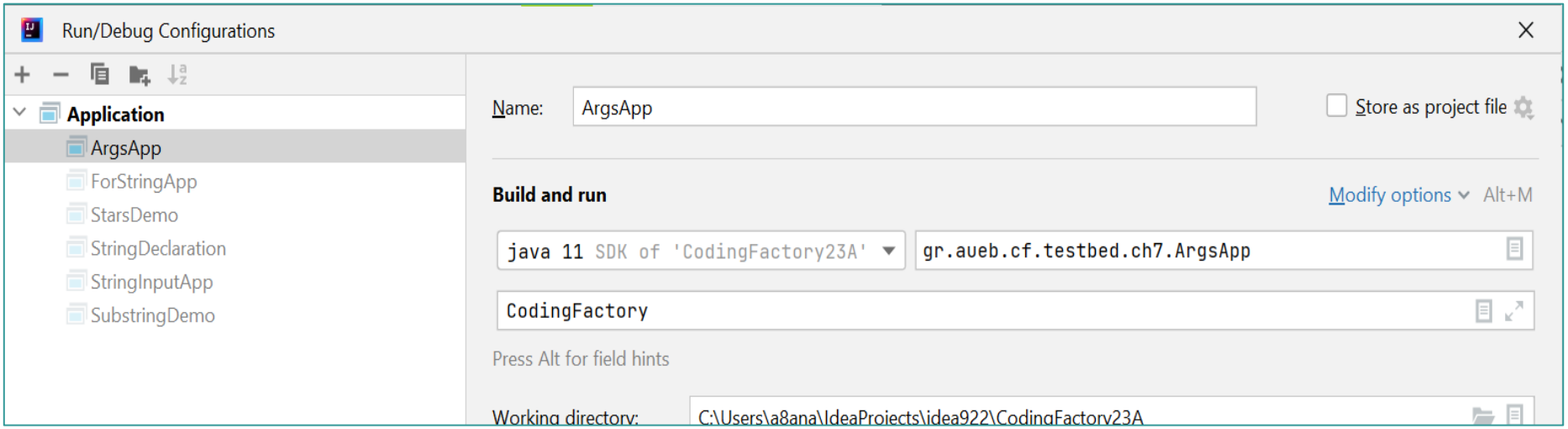
Παράμετροι γραμμής εντολών

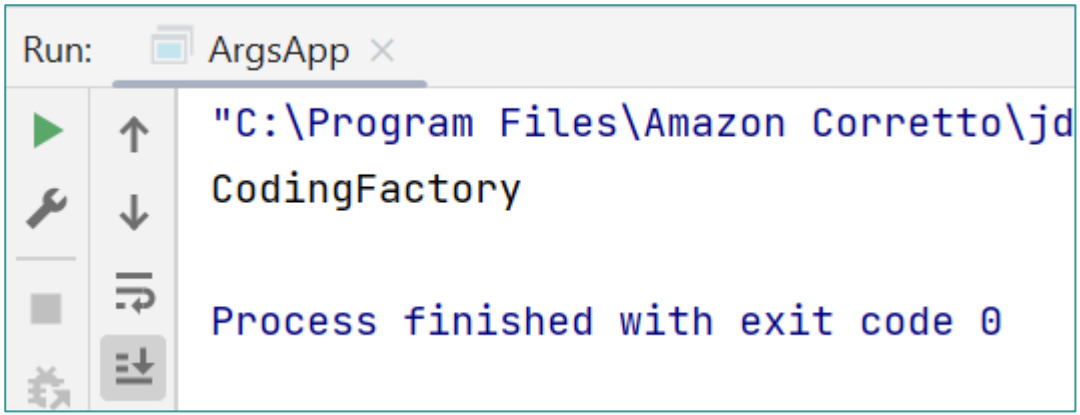
- Τα `args[0]`, `args[1]`, …, `args[n]` αντιστοιχούν στην $1^η$, $2^η$, …, $ν^η$ παράμετρο αντίστοιχα
- `args.length` είναι το πλήθος των παραμέτρων

Args


```java
package gr.aueb.cf.testbed.ch7;

/**
 * Prints the first CLI argument.
 */
public class ArgsApp {

    public static void main(String[] args) {
        if (args.length != 1) return;
        System.out.println(args[0]);
    }
}
```

Run / Edit Configurations





Μικρή εργασία κεφαλαίου

- Δημιουργήστε δύο προγράμματα, ένα για ***κρυπτογράφηση*** και ένα για ***αποκρυπτογράφηση***
  - Κρυπτογράφηση: Ένα πρόγραμμα που να κρυπτογραφεί / ***αντικαθιστά κάθε χαρακτήρα του String με τον λεξικογραφικά (ASCII-κογραφικα!) επόμενο***
  - Αποκρυπτογράφηση: Ένα πρόγραμμα που να αποκρυπτογραφεί / ***αντικαθιστά κάθε χαρακτήρα του String με τον λεξικογραφικά (ASCII-κογραφικα!) προηγούμενο***