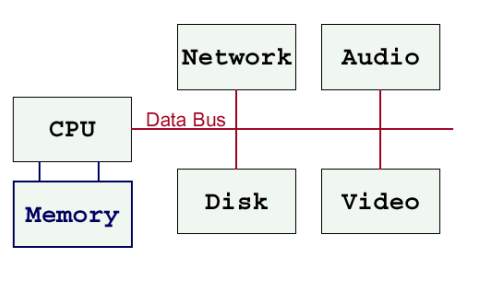
Διαχείριση Μνήμης

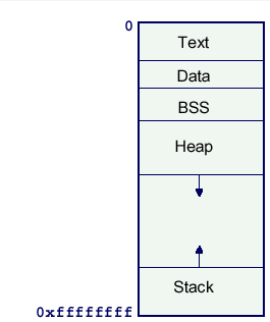
Δέσμευση Μνήμης

- Ο καλός προγραμματισμός επιβάλλει την αποδοτική χρήση της μνήμης του Η/Υ.
- Είναι σημαντικό να καταλαβαίνουμε τις διαδικασίες δέσμευσης μνήμης:
  - Για την αποδοτική δέσμευση δομών δεδομένων μη προκαθορισμένου μεγέθους.
  - Για την αποφυγή «διαρροών μνήμης» (memory leaks).
  - Για την επίδοση του εκτελούμενου προγράμματος (run-time performance).

Μνήμη προγραμμάτων

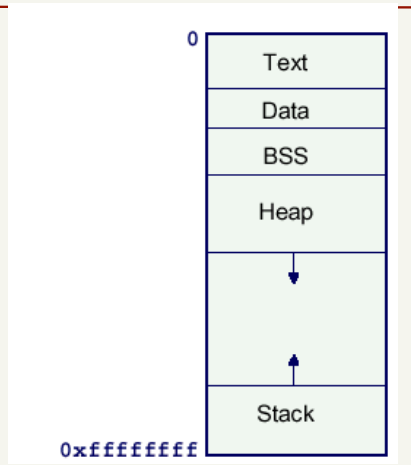
- Χώρος αποθήκευσης μεταβλητών, δεδομένων, κώδικα, κλπ.
- Το UNIX παρέχει στις διεργασίες μας έναν εικονικό χώρο διευθύνσεων (virtual address space).





Σχεδιάγραμμα της μνήμης μιας διεργασίας

- Text: δυαδικός κώδικας προγράμματος της διεργασίας.
- Data: σταθερές προγράμματος.
- BSS (Block Started by Symbol (BSS): γενικές και στατικές μεταβλητές (most modern assemblers produce a BSS section in their output object module containing all reserved uninitialized space).
- Stack: στοίβα, τοπικές μεταβλητές.
- Heap: δυναμική μνήμη.

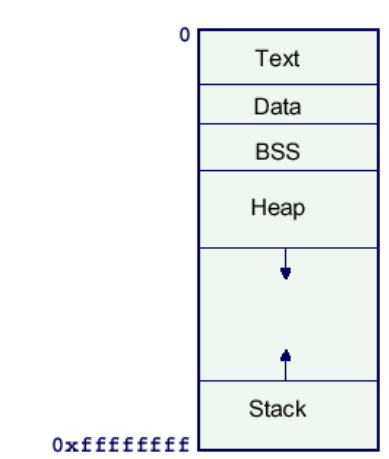


Σχεδιάγραμμα μνήμης διεργασιών

ΕΠΛ233

- Στην Γ/Π C:

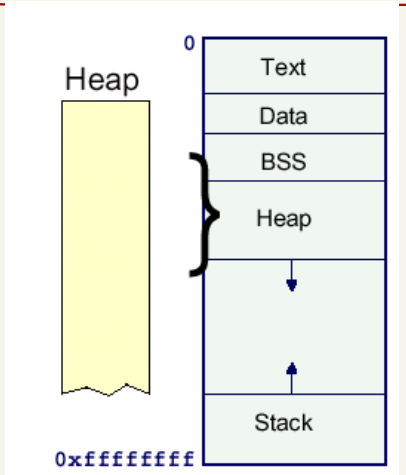
```
int iSize;              ← BSS
char *f(void) {

    char *p;            ← stack
    iSize = 8;          ← data
    p = malloc(iSize);  ← heap

    return p;
}
```



Δέσμευση Μνήμης

ΕΠΛ233

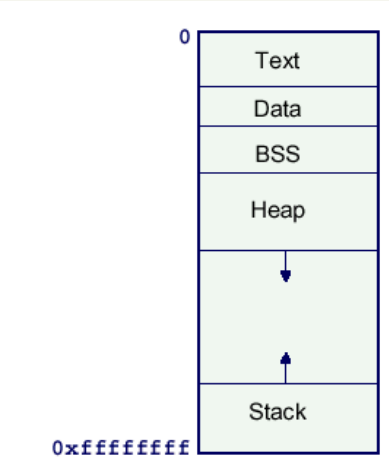
- Οι γενικές και οι στατικές μεταβλητές κατά την αρχικοποίηση του προγράμματος.
- Οι τοπικές μεταβλητές κατά την κλήση μιας υπορουτίνας/ μεθόδου.
- Η δυναμική μνήμη με τη χρήση της malloc (στη C) και με τη δημιουργία νέων αντικειμένων (JAVA).



Αποδέσμευση Μνήμης

ΕΠΛ233

- Γενικές και στατικές μεταβλητές με τη λήξη ενός προγράμματος.
- Τοπικές μεταβλητές με την επιστροφή από μια υπορουτίνα/ μέθοδο.
- Δυναμική μνήμη με χρήση της free (στη C) και από τον αποκομιστή σκυβάλων (garbage collector) στη JAVA.
- Όλη η μνήμη αποδεσμεύεται με τη λήξη του προγράμματος.



Αποθηκευτικοί Χώροι

ΕΠΛ233

- **Καταχωρητές** (Registers):
  - Μέσα στον επεξεργαστή
  - Γρήγορη πρόσβαση
  - Μικρός αριθμός
  - Διαφανείς προς τον προγραμματιστή
- **Στοίβα** (Stack):
  - Τοποθετείται στον εικονικό χώρο διευθύνσεων μνήμης (RAM)
  - Απευθείας πρόσβαση από τον επεξεργαστή μέσω του **Δείκτη Στοίβας (Stack Pointer)**
  - Χρησιμοποιείται για δεδομένα για τα οποία ο μεταφραστής-JAVA μπορεί να γνωρίζει ακριβές μέγεθος και χρόνο ζωής (ώστε να διαχειρίζεται κατάλληλα τον Δ/Σ).
  - Πολύ γρήγορη και αποδοτική πρόσβαση.
  - Αποθηκεύει **εγγραφήματα δραστηριοποίησης** (activation records ή stack frames).
  - Περιορισμένης λειτουργικότητας, γι' αυτό και χρησιμοποιείται κυρίως για την αποθήκευση ορισμένων μόνο δεδομένων και κυρίως χειριστήριων αντικειμένων.

Αποθηκευτικοί Χώροι

- **Σωρός** (heap):
  - Τοποθετείται στον εικονικό χώρο διευθύνσεων μνήμης (RAM)
  - Μνήμη Γενικής χρήσης, όπου «διαβιούν» όλα τα αντικείμενα της JAVA
  - Πιό ευέλικτος από την στοίβα: ο μεταφραστής δεν χρειάζεται να γνωρίζει εκ των προτέρων πόση μνήμη να δεσμεύσει στον σωρό για την αποθήκευση αντικειμένων, ή τον χρόνο ζωής των καταχωρούμενων αντικειμένων.
  - Όποτε χρειαστεί να δημιουργηθεί ένα αντικείμενο, πρέπει να γραφτεί ο σχετικός κώδικας για την δημιουργία του και να κληθεί με την χρήση της **new**. Τότε γίνεται η δέσμευση της μνήμης και εκτελείται ο κώδικας αρχικοποίησης.
  - Πιό αργή η καταχώρηση και πρόσβαση απ' ότι στην στοίβα.

Η αποθήκευση των δεδομένων

- **Στατική Αποθήκευση (static storage)**
  - Η στατική αποθήκευση υποδηλώνει την αποθήκευση δεδομένων σε συγκεκριμένη-αμετάβλητη θέση (fixed location).
  - Η πραγματική αποθήκευση στατικών δεδομένων γίνεται στη RAM.
  - Περιέχει δεδομένα τα οποία είναι διαθέσιμα για όλη την διάρκεια εκτέλεσης του προγράμματος.
  - Η λέξη-κλειδί static μπορεί να χρησιμοποιηθεί για τον προσδιορισμό κάποιου στοιχείου μιας κλάσης ως στατικού.
  - Τα αντικείμενα στην JAVA δεν τοποθετούνται ποτέ στην στατική μνήμη.
- **Σταθερή Μνήμη (constant storage)**
  - Αντιστοιχεί στις σταθερές τιμές που χρησιμοποιούνται στον κώδικα.

Αποθήκευση εκτός RAM

- Αφορά σε δεδομένα τα οποία «διαβιούν» τελείως εκτός του προγράμματος, ενώ το πρόγραμμα δεν τρέχει.
- Εκτός ελέγχου από το πρόγραμμα.
- Υπάρχουν δύο βασικά παραδείγματα:
  - **Streamed objects:** αντικείμενα που μετατρέπονται σε ροή χαρακτήρων (byte stream) συνήθως για να σταλούν σε μιάν άλλη μηχανή.
  - **Persistent objects:** αντικείμενα που τοποθετούνται στον δίσκο ώστε να διατηρήσουν την κατάστασή τους όταν το πρόγραμμα τελειώσει.
- Βασικό χαρακτηριστικό αυτών των αντικειμένων είναι ότι μπορούν να "αναστηθούν" σε κανονικά αντικείμενα της RAM, όταν χρειασθεί.

Αντικείμενα, Κλάσεις, Αρχέγονοι Τύποι, Μέθοδοι

Αντικείμενα στη JAVA

ΕΠΛ233

- Τα δεδομένα στην JAVA κωδικοποιούνται σαν αντικείμενα.
- Για κάθε αντικείμενο πρέπει να υπάρχει ένα **χειριστήριο** (handle) μέσω του οποίου αποκτούμε πρόσβαση στο ίδιο το αντικείμενο και το διαχειριζόμαστε (τις μεθόδους του, τα δεδομένα του).
- Π.χ.

```
String s; // δημιουργία χειριστηρίου

String s = new String("asdf");
// αρχικοποίηση αντικειμένου
```


Δημιουργία αντικειμένων

ΕΠΛ233

- Μετά την δημιουργία-δήλωση ενός χειριστηρίου, πρέπει να γίνει η "διασύνδεσή" του με κάποιο νέο αντικείμενο.
- Η δημιουργία ενός νέου αντικειμένου γίνεται με τη χρήση της εντολής **new:**

```
String s = new String("asdf");
      // String είναι ειδική περίπτωση τύπου
String s = "asdf";

BankAcct b1 = new BankAcct(21338, 0.0);
```

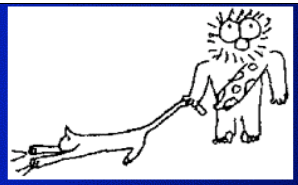

Αρχέγονοι Τύποι-primitive types

ΕΠΛ233

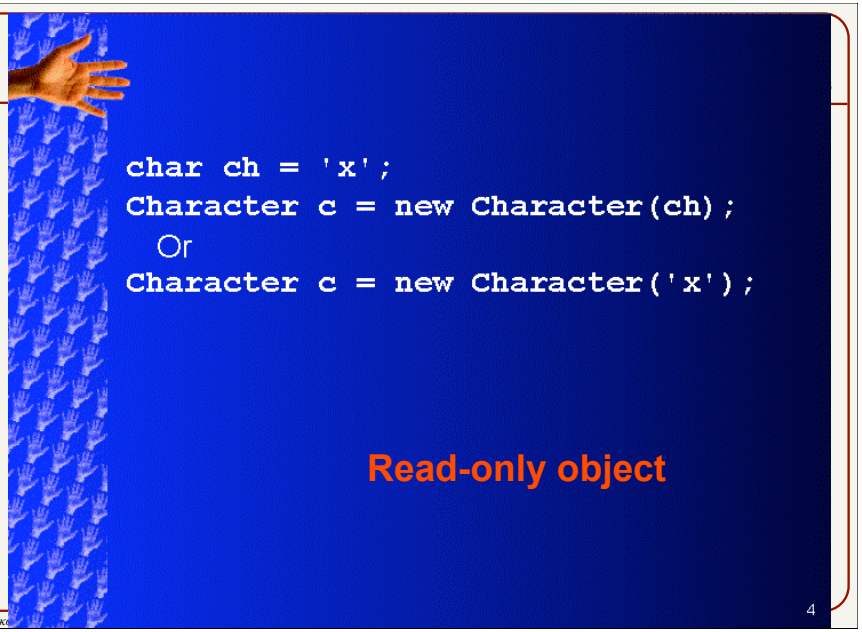
- Η δημιουργία αντικειμένων με την **new** έχει κόστος, λόγω της αποθήκευσης στον σωρό και ό,τι αυτή συνεπάγεται.
- Για απλούς-αρχέγονους τύπους δεδομένων, η JAVA ξεφεύγει από τον μοντέλο των αντικειμένων και ακολουθεί την προσέγγιση της C και της C++.
  - Έτσι, με την δήλωση μιας τοπικής μεταβλητής ως έχουσας αρχέγονο τύπο, "δημιουργείται" αυτόματα χώρος για την μεταβλητή αυτή - χωρίς χρήση της **new** - και η μεταβλητή τοποθετείται στην στοίβα.
  - Τι γίνεται για πεδία δεδομένων αρχέγονων τύπων;
- Ο ακριβής χώρος που κρατιέται για μεταβλητές αρχέγονων τύπων καθορίζεται από τον μεταφραστή της JAVA.
- Υπάρχουν ωστόσο και κλάσεις-συσκευαστές (wrapper classes) για τους αρχέγονους τύπους:

```
char c = 'x';
Character C = new Character('x');
```


Primitives



- Built-in types: *not* object references, but variables on the stack like C. **boolean**, **char** (Unicode), **byte**, **short**, **int**, **long**, **float**, **double**
- Same operations as C/C++, same syntax
- Size of each data type is machine independent!
- Portability & performance implications
- To create objects, wrapper classes are provided: **Boolean**, **Character**, **Byte**, **Short**, **Integer**, **Long**, **Float**, **Double**. Read only!



Πεδίο ισχύος - Εμβέλεια (scoping)

ΕΠΛ233

- Πεδίο ισχύος μεταβλητών (scoping). Μια μεταβλητή που ορίζεται μέσα σε κάποιο πεδίο, είναι υπαρκτή μέχρι το τέλος του πεδίου αυτού.

```
{
  int x = 12;
  {
      int q = 96;
  }
}
```

- Σε αντίθεση με την C και την C++, το ακόλουθο είναι συντακτικό σφάλμα:

```
{
  int x = 12;
  {
      int x = 96;
  }
}
```


Χρόνος ζωής αντικειμένων

ΕΠΛ233

- Ένα αντικείμενο που δημιουργείται με χρήση του **new** διατηρείται ακόμη και μετά το τέλος του σχετικού πεδίου εμβέλειας (scope).
- Το αντίστοιχο, ωστόσο, χειριστήριο, εξαφανίζεται:

```
{
  String s = new String("a string");
  /* end of scope */
}
```

- Αποκομιστής σκυβάλων (garbage collector)

Ο προγραμματιστής δεν καταστρέφει αντικείμενα!

Δημιουργία νέων τύπων: κλάσεις

ΕΠΛ233

- Ο τύπος των αντικειμένων στην JAVA καθορίζεται με τον ορισμό κλάσεων. Π.χ:

```
class ATypeName { /* class body */}
```

- Αντικείμενα μιας κλάσης δημιουργούνται με χρήση της **new:**

```
ATypeName a = new ATypeName();
```

- Στον ορισμό των κλάσεων συμπεριλαμβάνονται:
  - Πεδία δεδομένων (fields):
    - Αντικείμενα οποιουδήποτε τύπου, που αντιπροσωπεύονται από κάποια χειριστήρια. Χρήζουν αρχικοποίησης με την **new.**
    - Μεταβλητές αρχέγονων τύπων.
  - Μέθοδοι
- Κάθε αντικείμενο διατηρεί την δική του μνήμη για τα πεδία δεδομένων του. Τα δεδομένα ενός αντικειμένου **δεν** μπορούν να διαμοιρασθούν με άλλα αντικείμενα.

Προκαθορισμένες Τιμές Πεδίων

- Αν δεν κάνουμε εμείς ρητή αρχικοποίηση των μελών αρχέγονου τύπου μιας κλάσης, ο μεταγλωττιστής δίνει τις δικές του προκαθορισμένες τιμές, ως εξής:
- Για boolean μεταβλητές: **false**
- Για άλλους αρχέγονους τύπους το **0.**
- Για χειριστήρια αντικειμένων: **null**

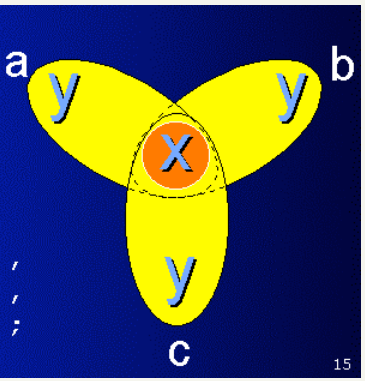
Στατικότητα

Η χρήση του static

- Για την πρόσβαση σε δεδομένα και μεθόδους προϋποτίθεται η δημιουργία σχετικού αντικειμένου.
- Υπάρχουν δύο περιπτώσεις όπου αυτό **δεν** είναι επιθυμητό:
  - Αν θέλουμε να χρησιμοποιείται μόνο μια θέση αποθήκευσης για κάποιο συγκεκριμένο δεδομένο, ανεξάρτητα του πόσα αντικείμενα δημιουργούνται.
  - Αν χρειαζόμαστε μεθόδους που δεν αντιστοιχούν σε κάποιο συγκεκριμένο αντικείμενο. Δηλ., αν θέλουμε να καλούμε τη μέθοδο χωρίς να έχουμε δημιουργήσει αντικείμενα.
- Οι δυνατότητες αυτές μπορούν να υλοποιηθούν με την χρήση της λέξης κλειδί **static.**
- Δεδομένα και μέθοδοι που ορίζονται σαν **static** αποκαλούνται και class-data, class-methods.

Η χρήση του static


```
class WithStaticData {
  static int x;
  int y;
}

WithStaticData
  a = new WithStaticData(),
  b = new WithStaticData(),
  c = new WithStaticData();
```



Static (παραδείγματα)


```
class StaticTest {
        static int i = 47;
}
StaticTest st1 = new StaticTest();
StaticTest st2 = new StaticTest();
```

- st1.i και st2.i δείχνουν στην ίδια τιμή.
- Αναφορά σε στατικές μεταβλητές:
  - είτε μέσω αντικειμένου
  - είτε μέσω κλάσης: StaticTest.i ++;

```
class StaticFun {
        static void incr() {StaticTest.i ++; }
}
StaticFun sf = new StaticFun();
sf.incr();
StaticFun.incr();
```


Παράδειγμα δέσμευσης μνήμης


```
public class ChargeClient {

   public static void main(String[] args) {
        double x = Double.parseDouble(args[0]);
        double y = Double.parseDouble(args[1]);

        Charge c1 = new Charge(.51, .63, 21.3);
        Charge c2 = new Charge(.13, .94, 81.9);

        double v1 = c1.potentialAt(x, y);
        double v2 = c2.potentialAt(x, y);
        StdOut.println(v1+v2);
    }
}
```


Παράδειγμα χρήσης static

- Πώς μπορούμε να κρατάμε ένα μετρητή των αντικειμένων τύπου φορτίο που δημιουργούνται;

Υλοποίηση Κλάσης Charge


```
public class Charge {
 private double rx, ry;    // position
 private double q;         // charge
 static int counter=0;     // charge counter

 public Charge(double x0, double y0, double q0) {
     rx = x0;
     ry = y0;
     q  = q0;
     counter += 1;
 }
 public double potentialAt(double x, double y) {
        ...
 }

 public String toString() {
     return q + "number " + counter +
          " at " + "(" + rx + ", " + ry + ")";
 } }
```

Χρήση στατικών μεθόδων (class methods)


```
public class LeapYear {

   public static boolean isLeapYear(int year) {
      boolean isLeapYear;

      // divisible by 4
       isLeapYear = (year % 4 == 0);

      // divisible by 4 and not 100
       isLeapYear = isLeapYear && (year % 100 != 0);

      // divisible by 4 and not 100 unless divisible by 400
       isLeapYear = isLeapYear || (year % 400 == 0);

       return isLeapYear;
   }
}
```


Βρείτε αν ένα έτος είναι δίσεκτο


```
public class TestLeapYear {

  public static void main(String[] args) {

      int year = Integer.parseInt(args[0]);

      LeapYear leapYearCheck = new LeapYear();

      System.out.println(LeapYear.isLeapYear(year));
  }
}
```


Βρείτε τυχαίο ακέραιο μεταξύ 0 - N


```
public class RandomInteger {
 public static int printRandInt(int max) {

    // a pseudo-random real between 0.0 and 1.0
    double r = Math.random();

    // a pseudo-random integer between 0 and N-1
    int n = (int) (r * max);

    return n;
 }
}
```