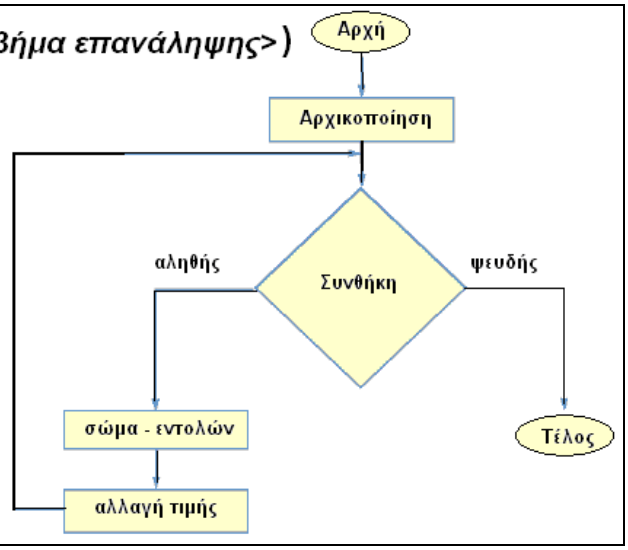
Εντολές επανάληψης ή βρόχων

Μία από τις σημαντικότερες δυνατότητες του προγραμματισμού είναι και η δυνατότητα εκτέλεσης εντολής ή μπλοκ εντολών για ένα οποιοδήποτε αριθμό επαναλήψεων. Η δυνατότητα αυτή υλοποιείται με τις **εντολές επανάληψης** ή **βρόχων** *(loops).* Οι εντολές επανάληψης της Java είναι οι: **for, while** και **do - while**. Αυτές οι εντολές δημιουργούν βρόχους που περικλείουν εντολές οι οποίες εκτελούνται μέχρι να υλοποιηθεί μία συνθήκη τερματισμού.

Η εντολή - for

Είναι η εντολή που εκτελείται για προκαθορισμένο αριθμό επαναλήψεων. Η for ελέγχεται από ένα ***μετρητή*** ο οποίος μεταβάλλεται από μία ***αρχική τιμή*** μέχρι να ξεπεράσει μία ***τελική τιμή.*** Στην εντολή καθορίζεται επίσης και το βήμα μεταβολής του μετρητή για την κάθε επανάληψη. Η γενική της μορφή είναι:

```
for (<αρχική τιμή> ; <συνθήκη τερματισμού> ; <βήμα επανάληψης>)
 {
  //σώμα εντολών
 }
```



Παραδείγματα:

Επανάληψη της θηλιάς για **10 – φορές**.

1) for (int i=1; i<=10; i++) System.out.println(i);

Ο βρόχος επαναλαμβάνεται όσο ικανοποιείται η συνθήκη. Στην πρώτη φάση της αρχικοποίησης η μεταβλητή i, που ισχύει μόνο μέσα στην for, λαμβάνει τιμή 1. Αμέσως ελέγχεται η συνθήκη τερματισμού (i<=10) και επειδή το i ικανοποιεί την συνθήκη η εντολή εκτελείται. Ο μετρητής αυξάνει κατά 1 και ελέγχεται ξανά η συνθήκη τερματισμού. Επειδή ικανοποιείται η συνθήκη ο βρόχος επαναλαμβάνεται και η όλη διαδικασία επαναλαμβάνεται μέχρι το i να πάρει την τιμή 11, οπότε παύει να ικανοποιείται η συνθήκη και ο βρόχος τερματίζεται. Έτσι η εντολή System.out.println(i); εκτελείται συνολικά 10 φορές. Το ίδιο αποτέλεσμα θα πάρουμε με τις παρακάτω εντολές for (όμως με διαφορετική σύνταξη):

2) for (int i=0; i<10; i++) System.out.println(i);

3) for (int i=10; i>0; i--) System.out.println(i);

```
class Forloop {
  public static void main(String args[]) {
        int i;

        for(i=10; i>0; i--)
              System.out.println("Number : " + i);
  }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα:

```
Number :  10
Number :  9
Number :  8
Number :  7
Number :  6
Number :  5
Number :  4
Number :  3
Number :  2
Number :  1
```

Στα τρία αυτά παραδείγματα ορίζουμε την μεταβλητή του βρόχου μέσα στον ίδιο τον βρόχο. Συνήθως ο ορισμός της γίνεται έξω από την for και μόνο η αρχικοποίησή της γίνεται στην for. Δηλαδή:

```
int i;
for(i=10; i>0; i--)
    System.out.println("Number : " + i);
```

Στο παρακάτω παράδειγμα η συνθήκη τερματισμού είναι boolean και όχι συνθήκη τιμής.

```
boolean ok = false;
for (int i = 1; ! ok; i++)
{
  // …
  if (metr >10) ok = true;
       :       :
 }
```

Στο παρακάτω παράδειγμα θα χρησιμοποιήσουμε βρόχο όπου όμως η τιμή εκκίνησης και το βήμα της επανάληψης ορίζονται έξω από αυτόν.

```
class ForVar {
  public static void main(String args[]) {
    int i;
    boolean done = false;
    i = 0;

    for( ; !done; ) {
      System.out.println("to i einai " + i);
      if(i == 10) done = true;
      i++;
    }
  }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα:

```
to i einai 0
to i einai 1
to i einai 2
to i einai 3
to i einai 4
to i einai 5
to i einai 6
to i einai 7
to i einai 8
to i einai 9
to i einai 10
```

Όταν δεν ορίσουμε τα τρία μέρη του βρόχου, τότε ο βρόχος αυτός θα είναι ατέρμονος (χωρίς τέλος). Για παράδειγμα:

```
for (  ; ; ) {
  //..
}
```

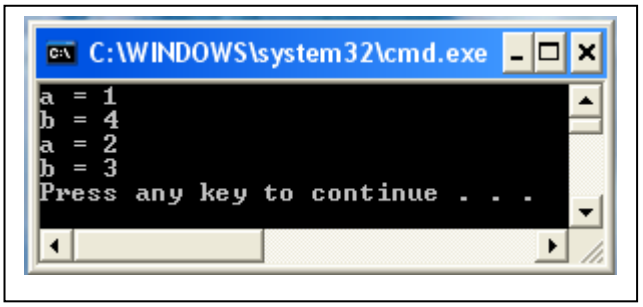
Ο διαχωριστής κόμμα (,) στην εντολή - for

Σε ένα βρόχο for μπορούμε να συμπεριλάβουμε πολλαπλές αρχικοποιήσεις μεταβλητών διαχωρισμένων με comma (,). Προσοχή όμως στη χρήση του γιατί γίνονται εύκολα λάθη. Ένα παράδειγμα με τη χρήση του διαχωριστή κόμμα είναι όταν δίνουμε αρχικές τιμές στις μεταβλητές του βρόχου. Έτσι αν είχαμε τον παρακάτω κώδικα:

```
class Sample {
  public static void main(String args[]) {
    int a, b;

    b = 4;
    for(a=1; a<b; a++) {
      System.out.println("a = " + a);
      System.out.println("b = " + b);
      b--;
    }
  }
}
```

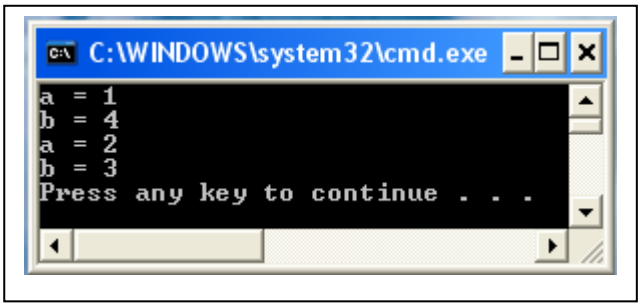
Με αποτέλεσμα:



θα μπορούσαμε να γράψουμε τις τιμές b = 4 και b -- μέσα στον ορισμό της for:

```
class Comma {
  public static void main(String args[]) {
    int a, b;

    for(a=1, b=4; a<b; a++, b--) {
      System.out.println("a = " + a);
      System.out.println("b = " + b);
    }
  }
}
```



Εστιασμένες ή φωλιασμένες - for

Εντολές for που εκτελούνται μέσα σε άλλες εντολές – for. Η λειτουργία ενός φωλιασμένου for : ***για κάθε τιμή του εξωτερικού βρόχου, ο εσωτερικός εκτελείται τόσες φορές όσες ορίζονται από τον μετρητή του και την συνθήκη τερματισμού ή αλλιώς ο εσωτερικός βρόχος επαναλαμβάνεται από την αρχή μέχρι το τέλος για κάθε μια επανάληψη του εξωτερικού βρόχου.***

Ένα παράδειγμα φωλιασμένων - for θα δούμε στο παρακάτω παράδειγμα. Στο παράδειγμα αυτό για i=0 το j (και ο βρόχος) θα εκτελεστεί 10 φορές, για i=1 το j θα εκτελεστεί 9 φορές, κ.ο.κ.

```
class ForNested {
  public static void main(String args[]) {
    int i, j;

    for(i=0; i<10; i++) {
      for(j=i; j<10; j++)
        System.out.print("*");
        System.out.println();
    }
  }
}
```

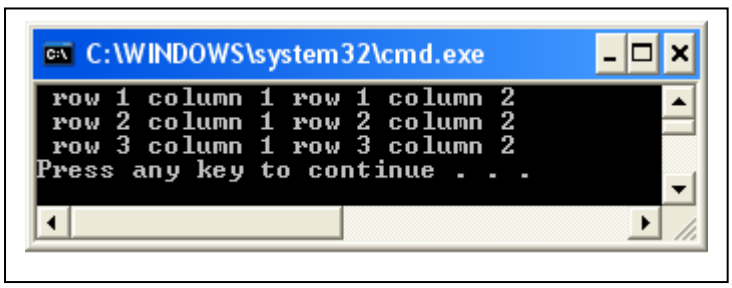
Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα:

```
**********
*********
********
*******
******
*****
****
***
**
*
```

Παράδειγμα 2ο

```
class ForNested1 {
  public static void main(String args[]) {
   int rowNum, columnNum;
   for(rowNum=1; rowNum<=3; rowNum++)    {
     for(columnNum=1; columnNum<=2; columnNum++)
       System.out.print(" row " + rowNum +  " column " + columnNum);
       System.out.println();
     }
  }
}
```

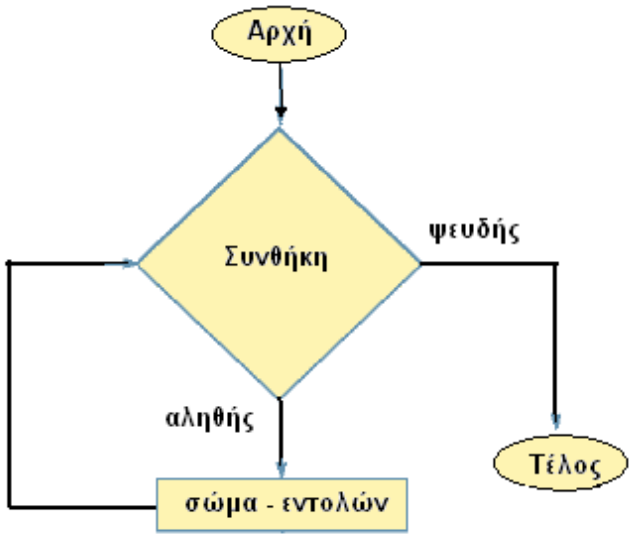
Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα:



Η εντολή - while

Η εντολή while χρησιμοποιείται για να επαναλαμβάνει την εκτέλεση μιας εντολής ή ενός μπλοκ εντολών με βάση την αποτίμηση μιας λογικής έκφρασης. Όσο η έκφραση είναι αληθής η εντολή ή το μπλοκ των εντολών εκτελείται. Όταν γίνει ψευδής παύει να λειτουργεί ο βρόχος και εκτελείται η επόμενη εντολή. Η γενική της μορφή είναι:

```
while <συνθήκη>
{
  //σώμα εντολών
}
```



Η συνθήκη, που βρίσκεται στην αρχή του βρόχου, ελέγχεται πριν εκτελεστεί η εντολή ή το σώμα των εντολών και αν το αποτέλεσμα είναι false, τότε δεν εκτελούνται καθόλου. Επομένως, για να λειτουργήσει ο βρόχος η συνθήκη θα πρέπει να έχει τιμή true από την αρχή. Σε κάθε επανάληψη ελέγχεται η συνθήκη και αν είναι true ο βρόχος εκτελείται, διαφορετικά τελειώνει και εκτελείται η επόμενη εντολή.

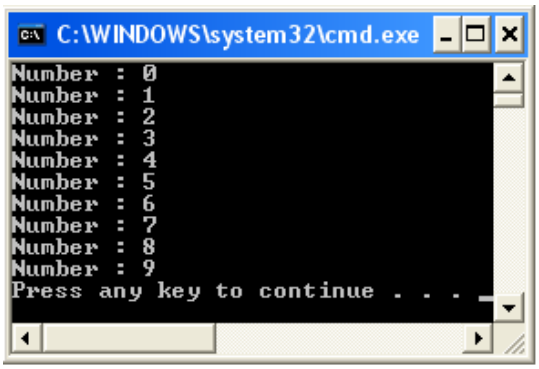
Στα παρακάτω παραδείγματα θα δούμε το βρόχο να εκτελείται 10 – φορές.

1) με αύξουσα μέτρηση (από 0 έως 9).

```
class DemoWhile {
  public static void main(String args[]) {
    int i = 0;

    while(i < 10) {
      System.out.println("Number : " + i);
      i++;
    }
  }
}
```

Τα αποτελέσματα είναι:

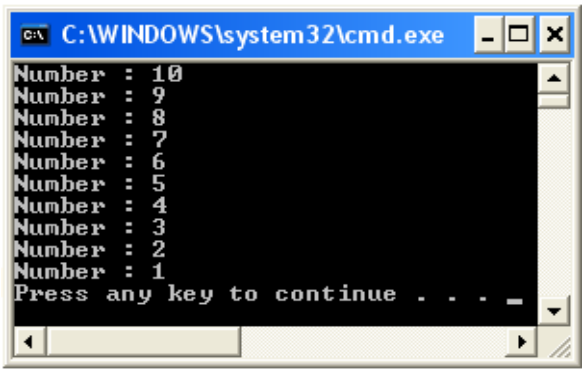


2) με φθίνουσα μέτρηση (από το 10 προς το 1).

```
class DemoWhile {
  public static void main(String args[]) {
    int i = 10;

    while(i > 0) {
      System.out.println("Number : " + i);
      i--;
    }
  }
}
```

Τα αποτελέσματα είναι:



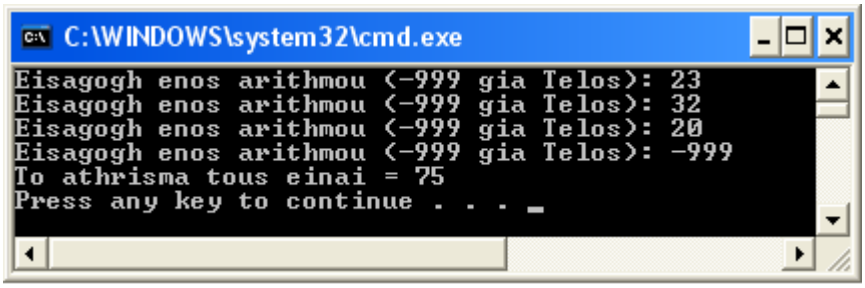
Παραδείγματα

Το παρακάτω πρόγραμμα εισάγει αριθμούς από το πληκτρολόγιο (μέχρι να εισαχθεί το -999), οπότε θα εμφανιστεί το άθροισμά τους.

```
import java.io.*;
class testWhile {
 public static void main(String[] args) throws IOException {
   int eisodos, sum=0;
   BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
   boolean t=true;
```

```
  while (t){
    System.out.print("Eisagogh enos arithmou (-999 gia Telos): ");
    eisodos = Integer.parseInt(br.readLine());
    if (eisodos!=-999) sum=sum+eisodos;
        else {System.out.println("To athrisma tous einai = " + sum);t=false;}
   }
  }
 }
```

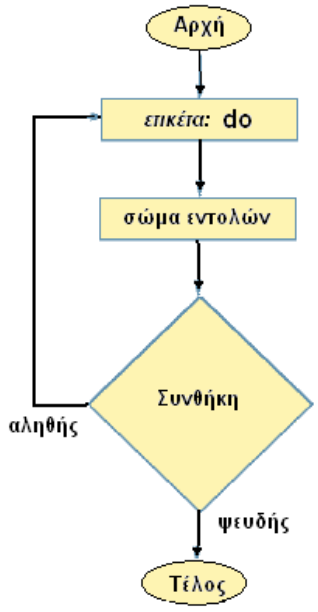
Αν εκτελέσουμε το πρόγραμμα ένα πιθανό αποτέλεσμα είναι:



Η εντολή do - while

Είναι η εντολή επανάληψης κατά την οποία το σώμα των εντολών θα εκτελεστεί τουλάχιστον μία φορά, καθότι η συνθήκη ελέγχεται στο τέλος του βρόχου. Δηλαδή, πρώτα εκτελείται το σώμα του βρόχου και μετά ελέγχεται η συνθήκη. Αν η τιμή της συνθήκης είναι αληθής, τότε θα συνεχιστεί η εκτέλεση του βρόχου, διαφορετικά θα τερματιστεί και θα εκτελεστεί η αμέσως επόμενη εντολή. Η γενική της μορφή είναι:

```
do {
   //σώμα βρόχου
} while <συνθήκη>;
```



Στο παρακάτω παράδειγμα θα δούμε όπως και στην while μία θηλιά να εκτελείται 10 – φορές με φθίνουσα τάξη : 10, 9, 8,….,1.

```
class DoWhile {
  public static void main(String args[]) {
    int i = 10;

    do {
      System.out.println("Number : " + i);
      i--;
    } while(i > 0);
  }
}
```

Στο ανωτέρω παράδειγμα η εντολή do – while θα μπορούσε να γραφεί και με τον παρακάτω τρόπο:

```
do
 System.out.println("Number : " + i);
while (--i > 0);
```

Παράδειγμα με τη χρήση και των τριών βρόχων

Το παρακάτω παράδειγμα συνοψίζει τη χρήση και των **τριών εντολών επανάληψης**. Δείχνει πως εκτελούνται οι τρεις διαφορετικοί βρόχοι για 10 επαναλήψεις (από το 0 έως το 9):

```
class testLoop {
 public static void main(String[] args){

  int i=0;

  while (i<10) {
     System.out.println(i);
     i++;
 }
  System.out.println("Telos While..");

  i=0;
  do {
          System.out.println(i);
  }while(++i<10);

   System.out.println("Telos do while..");

   for (i=0;i<10;i++) System.out.println(i);
   System.out.println("Telos for..");
   }
  }
```

Εντολές εξόδου

Είναι οι εντολές που τερματίζουν τους βρόχους. Οι εντολές αυτές είναι οι: **break, continue, return** και **exit**.

Η χρήση της εντολής break

Χρησιμοποιείται:

- στην εντολή switch για τον τερματισμό του κάθε case
- για τον τερματισμό ενός βρόχου
- αντί μίας 'εξευγενισμένης' – goto εντολής

Όταν εκτελεστεί η break μέσα σε ένα βρόχο, τότε ο βρόχος θα τερματιστεί και θα εκτελεστεί η αμέσως επόμενη εντολή.

Στο παρακάτω παράδειγμα θα δούμε τη χρήση της break μέσα σε ένα βρόχο. Ο βρόχος αντί να εκτελεστεί από το 0 έως το 99, όπως έχει αρχικά οριστεί, εκτελείται μόνο 10 – φορές.

```
class BreakLoop {
  public static void main(String args[]) {
    for(int i=0; i<100; i++) {
      if(i == 10) break;    // τερματίζει τη θηλιά ότα το i γίνει 10
      System.out.println("i: " + i);
    }
    System.out.println("End of the Loop");
  }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα:

```
i: 0
i: 1
i: 2
i: 3
i: 4
i: 5
i: 6
i: 7
i: 8
i: 9
End of the loop
```

Τα ίδια αποτελέσματα με τα ανωτέρω θα πάρουμε αν εκτελέσουμε τον παρακάτω κώδικα με την while :

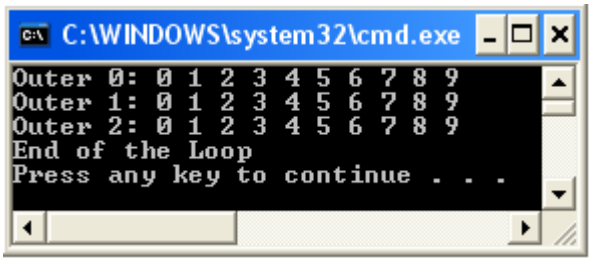
```
class BreakLoop2 {
  public static void main(String args[]) {
    int i = 0;
    while(i < 100) {
      if(i == 10) break;  // τερματίζει τη θηλιά ότα το i γίνει 10
      System.out.println("i: " + i);
      i++;
    }
    System.out.println("End of the Loop");
  }
}
```

Όταν η break βρίσκεται στον εσωτερικό βρόχο δύο φωλιασμένων βρόχων, προκαλεί τον τερματισμό του εσωτερικού βρόχου μόνο.

```
class BreakLoop3 {
  public static void main(String args[]) {
    for(int i=0; i<3; i++) {
      System.out.print("Outer " + i + ": ");


      for(int j=0; j<100; j++) {
        if(j == 10) break;    //τερματίζει τη θηλιά όταν το j γίνει 10
        System.out.print(j + " ");
      }
      System.out.println();
    }
    System.out.println("End of the Loop ");
  }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα:



Στα παρακάτω παραδείγματα θα δούμε τη χρήση της break σαν **goto** – εντολή μιας και δεν υπάρχει αυτή η εντολή στη java. Αυτό δεν σημαίνει ότι θα χρησιμοποιούμε την break σαν εντολή – goto. Προσέξτε την σύνταξη της εντολής **break <ετικέτα>.**

```
class Break {
  public static void main(String args[]) {
    boolean t = true;
    first: {
      second: {
        third: {
          System.out.println("Before the break");
          if(t) break second;        // τερματίζει το 2ο block
            System.out.println("This won't execute");
    }
     System.out.println("This won't execute");
     }
      System.out.println("After the break");
        }
   }
  }
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα:

```
Before the break
After the break
```

Η χρήση της εντολής – continue

Όταν εκτελεστεί η εντολή **continue**, παραλείπεται το υπόλοιπο κομμάτι κώδικα και η ροή του προγράμματος μεταφέρεται στην συνθήκη ελέγχου της θηλιάς.

Παραδείγματα:

Στο πρώτο πρόγραμμα χρησιμοποιούμε τον τελεστή % για να ελέγξουμε αν το i είναι ζυγό. Αν είναι ζυγό, τότε η θηλιά συνεχίζει χωρίς να εμφανίζει στην οθόνη νέα γραμμή.

```
class DemoContinue {
  public static void main(String args[]) {
    for(int i=0; i<10; i++) {
      System.out.print(i + " ");
      if (i%2 == 0) continue;
      System.out.println("");
    }
  }
}
```

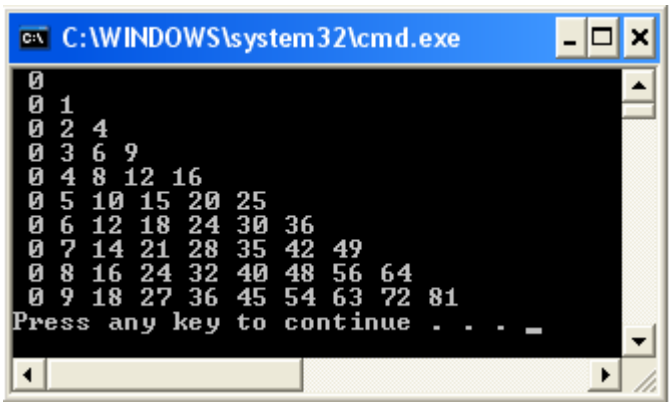
Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα:

```
0 1
2 3
4 5
6 7
8 9
```

Όπως και με τη break μπορούμε να χρησιμοποιήσουμε ετικέτα για να αλλάξουμε τη ροή του προγράμματος. Στο παρακάτω παράδειγμα υπολογίζεται ο τριγωνικός πίνακας πολλαπλασιασμού των αριθμών από το 0 έως το 9.

```
class ContinueLabel {
  public static void main(String args[]) {
outer: for (int i=0; i<10; i++) {
        for(int j=0; j<10; j++) {
          if(j > i) {
            System.out.println();
            continue outer;
          }
          System.out.print(" " + (i * j));
        }
      }
      System.out.println();
  }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα:



Η χρήση της εντολής – return

Η εντολή αυτή τερματίζει μία μέθοδο και επιστρέφει την ροή στο πρόγραμμα από το οποίο καλείται. Εννοείται ότι όταν η return εκτελεστεί στην main(), τότε η ροή του προγράμματος θα επιστρέψει στην έξοδο (γραμμή εντολών) όπως στο παρακάτω παράδειγμα:

```
class Return {
  public static void main(String args[]) {
    boolean t = true;
    System.out.println("Before the return");

    If(t) return;    //Επιστροφή στο καλόν πρόγραμμα
    System.out.println("Not shown");
  }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε το παρακάτω αποτέλεσμα:

```
Before the return
```

Όπως βλέπουμε η εντολή System.out.println("Not shown"); δεν εκτελείται. Μετά την εκτέλεση της return η ροή του προγράμματος επιστρέφει στη γραμμή εντολών.

Η χρήση της εντολής – exit

Η εντολή exit τερματίζει αμέσως την εκτέλεση του προγράμματος. Συντάσσεται με την System: **System.exit(0);** Η exit δέχεται ως όρισμα έναν ακέραιο, συνήθως χρησιμοποιούμε το 0 για να δηλώσουμε κανονικό τερματισμό του προγράμματος.