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

Ο τύπος δεδομένων boolean και οι δομές ελέγχου while και if

Αθ. Ανδρούτσος

Boolean και Συνθήκες Αλήθειας

- Ο ***τύπος boolean αποθηκεύει τιμές αλήθειας*** (true/false)
- Μία συνθήκη αλήθειας είναι συνήθως μία **σύγκριση** ανάμεσα σε δύο τιμές
- Για παράδειγμα η σύγκριση 12 > 5 είναι true
- Ενώ η σύγκριση 12 > 70 είναι false

Συνθήκες Αλήθειας

- Αν ισχύει επομένως μία συνθήκη αλήθειας τότε το αποτέλεσμα είναι μία τιμή αλήθειας που έχει την τιμή: **true**
- Αν δεν ισχύει μία συνθήκη αλήθειας τότε το αποτέλεσμα είναι: **false**

Μεταβλητές τύπου boolean


```
1   package gr.aueb.cf.ch3;
2
3   /**
4    * Δηλώσεις μεταβλητών boolean.
5    */
6   public class BoolApp {
7
8       public static void main(String[] args) {
9           boolean b;
10          boolean b1 = true, b2 = false;
11      }
12  }
```

- Δηλώσεις μεταβλητών boolean κάνουμε όπως και στις μεταβλητές int, ως:
- *boolean b;*
- Δήλωση και αρχικοποίηση κάνουμε ως:
- *boolean b1 = true;*
- boolean b2 = false;

Τελεστές

- Όπως στον τύπο int έχουμε τους αριθμητικούς τελεστές, έτσι και στον τύπο boolean έχουμε τους:
  - **Σχεσιακούς τελεστές**, και τους
  - **Λογικούς τελεστές**,

που θα δούμε στη συνέχεια

Σχεσιακοί Τελεστές

- ***Σχεσιακοί τελεστές***
  <, <=, >, >=, == (ίσο), != (διάφορο)
- Η ισότητα είναι **==** ώστε να διαφοροποιείται από τον τελεστή εκχώρησης =
- Η ανισότητα είναι **!=** όπου το ! διαβάζεται not, δηλαδή το != διαβάζεται not equal (not ίσο)

Υπολογισμός τιμών boolean

- Υπολογισμός τιμής boolean μέσω σύγκρισης
- Για παράδειγμα:
  - 12 > 1, 2 >= 2, -32 < -22 έχουν τιμή true
  - 13 != 13, 1 >= 2 έχουν τιμή false

Παραδείγματα - Άρτιοι

- (x % 2) == 0 , πότε είναι αληθής;
  - Όταν ο αριθμός x **είναι άρτιος (ή όταν δεν είναι περιττός)**
  - Οι άρτιοι διαιρούνται ακριβώς με το 2 και το x % 2 (x mod 2, το υπόλοιπο δηλαδή της διαίρεσης ενός αριθμού με το 2) για τους άρτιους είναι 0

Παραδείγματα – Περιττοί

- (x % 2) != 0 , πότε είναι αληθής;
  - Όταν ο αριθμός x **είναι περιττός**
  - Οι περιττοί δεν διαιρούνται ακριβώς με το 2 ενώ το x % 2 (x mod 2, το υπόλοιπο δηλαδή της διαίρεσης ενός αριθμού με το 2) για τους περιττούς δεν είναι 0, αλλά διάφορο από το 0

Boolean και Λογικές παραστάσεις

- Τιμές Boolean προκύπτουν και ως μία ακολουθία (λογική παράσταση) συγκρίσεων που ενώνονται με **λογικούς τελεστές** (Λογικό ΚΑΙ, Λογικό Ή, Λογικό ΟΧΙ)
- Για παράδειγμα:
  - (grade >= 5) ΚΑΙ (grade <= 10), επομένως μόνο εάν ο grade είναι 5, 6, 7, 8, 9, 10 η παράσταση (boolean expression) θα είναι αληθής
  - (age <= 18) ‘Η (age >= 67), επομένως μόνο αν το age είναι μικρότερο από 18 **ή** μεγαλύτερο από 67 το αποτέλεσμα θα είναι true

Λογικοί Τελεστές

- Όταν έχουμε περισσότερες από μία συνθήκες σύγκρισης ή μεταβλητές boolean και θέλουμε να τις συνενώσουμε σε μία παράσταση (boolean expression), τότε χρησιμοποιούμε τους ***τρεις (3) βασικούς λογικούς τελεστές***:
  - **&&** (Λογικό Και / AND)
  - **||** (Λογικό Ή / OR)
  - **!** (Λογικό Όχι / NOT)

Λογικό Και - Λογικό Ή

- Παρακάτω παρουσιάζονται οι πίνακες αλήθειας του Λογικού ΚΑΙ (&&) και Λογικού Ή (||) για κάθε τιμή (false ή true) των Α και Β
- Παρατηρούμε ότι το **Λογικό ΚΑΙ δίνει true μόνο αν Α == true και Β == true**, ενώ το **Λογικό Ή δίνει false μόνο αν Α == false και Β == false**

A	B	A && B	A || B
false	false	**false**	**false**
false	true	**false**	**true**
true	false	**false**	**true**
true	true	**true**	**true**

Έξοδος/Είσοδος τιμών boolean

- Έξοδος με print/println/printf
  - Εμφανίζεται true ή false
- Είσοδος με `Scanner.nextBoolean()`

Temperature App


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Αν η θερμοκρασία είναι < 0, τότε isTempBelowZero
 * γίνεται true, αλλιώς γίνεται false. Ο χρήστης δίνει
 * τη θερμοκρασία.
 */
public class TemperatureApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        boolean isTempBelowZero = false;
        int temp = 0;

        System.out.println("Please insert current temperature");
        temp = in.nextInt();

        isTempBelowZero = (temp < 0);

        System.out.println("Temperature is below zero: " + isTempBelowZero);
    }
}
```

- Μπορούμε να εκχωρούμε boolean παραστάσεις σε boolean μεταβλητές
- Η μεταβλητή (γρ. 20) *isTempBelowZero* παίρνει τιμή από το (temp < 0) που θα είναι true ή false ανάλογα με την τιμή θερμοκρασίας που θα δώσει ο χρήστης

Snowing App


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Αποφασίζει αν χιονίζει ή όχι, με βάση τη θερμοκρασία
 * αλλά και το αν βρέχει. Αν βρέχει και η θερμοκρασία
 * είναι < 0, τότε χιονίζει, αλλιώς όχι.
 */
public class SnowingApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        boolean isSnowing = false;
        boolean isRaining = false;
        int temp = 0;

        System.out.println("Please insert if it is raining (true/false)");
        isRaining = in.nextBoolean();
        System.out.println("Please insert temperature (int)");
        temp = in.nextInt();
        isSnowing = isRaining && (temp < 0);

        System.out.println("Is snowing: " + isSnowing);
    }
}
```

- Το Λογικό ΚΑΙ (&&) ανάμεσα σε δύο boolean τιμές θα δώσει αποτέλεσμα αληθές (true) αν isRaining **και** temp < 0, αλλιώς (αν είτε δεν isRaining ή δεν ισχύει temp < 0 ή και τα δύο) τότε θα δώσει false
- Η τιμή της παράστασης θα εκχωρηθεί στην isSnowing (γρ. 22)
- Οι παρενθέσεις είναι για την καλύτερη οργάνωση της παράστασης, όπως έχουμε αναφέρει για την χρήση παρενθέσεων. Οι τελεστές σύγκρισης έχουν μεγαλύτερη προτεραιότητα από τους λογικούς τελεστές, οπότε τεχνικά και χωρίς παρενθέσεις πρώτα θα εκτελεστεί η σύγκριση και μετά το λογικό &&

Short-Circuit


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Αποφασίζει αν χιονίζει ή όχι, με βάση τη θερμοκρασία
 * αλλά και το αν βρέχει. Αν βρέχει και η θερμοκρασία
 * είναι < 0, τότε χιονίζει, αλλιώς όχι.
 */
public class SnowingApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        boolean isSnowing = false;
        boolean isRaining = false;
        int temp = 0;

        System.out.println("Please insert if it is raining (true/false)");
        isRaining = in.nextBoolean();
        System.out.println("Please insert temperature (int)");
        temp = in.nextInt();
        isSnowing = isRaining && (temp < 0);

        System.out.println("Is snowing: " + isSnowing);
    }
}
```

- Και το λογικό ΚΑΙ (&&) και το λογικό Ή (||) είναι short-circuit

- Οι λογικοί τελεστές είναι **short-circuit**
- Δηλαδή, αν το *isRaining* είναι false δεν χρειάζεται να προχωρήσει η έκφραση στον υπολογισμό του (temp < 0) γιατί όποια και να είναι η τιμή του (true ή false) η τιμή της έκφρασης θα είναι false
- Θυμίζουμε ότι σε μία παράσταση με λογικό ΚΑΙ (&&) αν ένας από τους δύο τελεσταίους είναι false, τότε η παράσταση είναι false
- Αν ωστόσο το *isRaining* είναι true, θα πρέπει να προχωρήσει και ο υπολογισμός του (temp < 0)

Παράδειγμα – lights On (1)


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Αποφασίζει αν πρέπει να ανάψουν τα φώτα
 * ενός αυτοκινήτου με βάση τρεις μεταβλητές:
 * αν βρέχει ΚΑΙ ταυτόχρονα ισχύει ένα τουλάχιστον
 * από τα επόμενα: είναι σκοτάδι Η' τρέχουμε
 * (speed > 100). Τις τιμές αυτές τις λαμβάνουμε
 * από τον χρήστη (stdin).
 */
public class LightsOnApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        boolean isRaining = false;
        boolean isDark = false;
        boolean isRunning = false;
        int speed = 0;
        boolean lightsOn = false;
        final int MAX_SPEED = 100;
```

- Δηλώνουμε μεταβλητές και αρχικοποιούμε σε default τιμές
- Ως γενική αρχή, θα πρέπει να αρχικοποιούμε τις μεταβλητές που δηλώνουμε, σε αρχικές default τιμές, ώστε σε καμία στιγμή να μην έχουν απροσδιόριστες τιμές
- Για αυτό, δηλώνουμε και αρχικοποιούμε ταυτόχρονα
- Το IntelliJ δίνει warning αν αρχικοποιούμε και στη συνέχεια ξαναδίνουμε τιμή (μέσω Scanner), αλλά τα warnings δεν επηρεάζουν.

Παράδειγμα – lights On (2)


```
24          System.out.println("Please insert if it is raining (true/false)");
25          isRaining = in.nextBoolean();
26
27          System.out.println("Please insert if it is dark (true/false)");
28          isDark = in.nextBoolean();
29
30          System.out.println("Please insert car speed (int)");
31          speed = in.nextInt();
32
33          isRunning = (speed > MAX_SPEED);
34          lightsOn = isRaining && (isDark || isRunning);
35
36          System.out.println("Lights On: " + lightsOn);
37      }
38  }
```

- Διαβάζουμε τις τιμές που εισάγει ο χρήστης, υπολογίζουμε την παράσταση και εμφανίζουμε το αποτέλεσμα

Παράδειγμα – Δεξαμενές (1)


```java
package gr.aueb.cf.testbed.ch3;

import java.util.Scanner;

/**
 * Λαμβάνει true / false από το stdio
 * για τις δύο δεξαμενές αν είναι < 1/4
 * Επεξεργάζεται και ανάβει πορτοκαλί αν
 * η μία είναι <1/4 και κόκκινο αν και
 * οι δύο είναι <1/4.
 */
public class TankApp {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean isEmptyTank1 = false;
        boolean isEmptyTank2 = false;
        boolean isOrange = false;
        boolean isRed = false;

        System.out.println("Please insert tank status (true/false)");
        System.out.print("Tank 1 status: ");
        isEmptyTank1 = scanner.nextBoolean();

        System.out.print("Tank 2 status: ");
        isEmptyTank2 = scanner.nextBoolean();

        isOrange = isEmptyTank1 ^ isEmptyTank2;
        isRed = isEmptyTank1 && isEmptyTank2;

        System.out.println("Orange: " + isOrange + ", " + "Red: " + isRed);
    }
}
```

- Ακολουθείται και εδώ η ίδια λογική υπολογισμού των Boolean μεταβλητών isOrange, isRed μέσω λογικών παραστάσεων
- Ο τελεστής ^ υλοποιεί τη λογική XOR (δίνει true μόνο αν είναι true/false ή false/true) οπότε θα πάρουμε orange μόνο αν μία από τις δύο δεξαμενές είναι <1/4
- Red θα πάρουμε αν και τα δύο είναι true με τον τελεστή &&

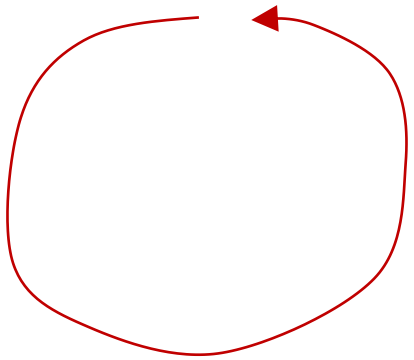
Δομές Ελέγχου (1)

- Μέχρι στιγμής έχουμε δει προγράμματα ως μία ακολουθία εντολών
- Τα περισσότερα προγράμματα όμως έχουν μία δομή επαναληπτική, δηλαδή **κάνουν το ίδιο πράγμα πολλές φορές**
- Για παράδειγμα, αν έχουμε ένα αρχείο φορολογουμένων, μπορούμε επαναληπτικά να υπολογίζουμε τον φόρο εισοδήματος για κάθε φορολογούμενο στο αρχείο, μέχρι να βρούμε τέλος αρχείου

Δομές Ελέγχου (2)

- Η λογική αυτή της επανάληψης (iteration) είναι πολύ σημαντική στον προγραμματισμό γιατί πολλές εφαρμογές έχουν αυτή τη λογική του WHILE-DO δηλαδή ΌΣΟ υπάρχουν δεδομένα/στοιχεία ΚΑΝΕ κάτι επαναληπτικά για κάθε στοιχείο

WHILE (ισχύει μία Boolean συνθήκη) DO



Η δομή ελέγχου while-do

- Η δομή ελέγχου ***while-do*** είναι **δομή επαναληπτική** (το do δεν το γράφουμε όπως βλέπουμε παρακάτω αλλά το εννοούμε). Συντάσσεται ως εξής:
- ```
  while (παράσταση αληθείας)
  {
      εντολή;
      εντολή;
      …..
      update την παράσταση αλήθειας
  }
  ```
- Πρώτα αποτιμάται η παράσταση αλήθειας (συνθήκη εισόδου αλλά και εξόδου). Όσο είναι αληθής (true) εκτελούνται οι εντολές (το σώμα της while), μέχρι να γίνει ψευδής (false) οπότε και σταματάει. Μέσα στο σώμα της while πρέπει να γίνεται update η παράσταση αλήθειας, ώστε κάποια στιγμή να σταματήσει να εκτελείται η while, αλλιώς θα εκτελείται για πάντα (αέναο loop)
  ```

Η εντολή while (1)

- Επαναληπτική δομή ελέγχου – οι επαναλήψεις ονομάζονται loops
- Αν την πρώτη φορά η συνθήκη εισόδου είναι false, τότε οι εντολές της while δεν εκτελούνται και το πρόγραμμα συνεχίζει από την επόμενη εντολή που ακολουθεί την while
- Γενικά η λογική της while είναι λογική ότι κάτι εκτελείται επαναληπτικά χωρίς να γνωρίζουμε το πλήθος των φορών που θα γίνει η επανάληψη, και θα τερματιστεί όταν διαβάσουμε μία ειδική τιμή (sentinel value)
- Ωστόσο μπορεί να γίνει και *κατάχρηση* της while και να εφαρμοστεί και σε περιπτώσεις που γνωρίζουμε από πριν πόσες φορές θα εκτελεστεί

Η εντολή while (2)

- Οι εντολές εκτελούνται μηδέν ή περισσότερες φορές
- Η συνθήκη εισόδου είναι και συνθήκη εξόδου, οπότε, όπως αναφέραμε, μία από τις εντολές της while πρέπει τελικά να αλλάζει την τιμή της συνθήκης σε false, αλλιώς θα έχουμε **ένα**η επανάληψη

Αστεράκια (1) – count up


```java
package gr.aueb.cf.ch3;

/**
 * Εμφανίζει 10 οριζόντια αστεράκια,
 * δηλαδή 1 αστεράκι 10 φορές.
 */
public class Stars10App {

    public static void main(String[] args) {

        int i = 1;

        while (i <= 10) {
            System.out.print("*");
            i++;
        }
    }
}
```

- Η while **εκτελείται 10 φορές** από i = 1 έως i = 10. Εκτελείται δηλαδή όσο το i είναι μικρότερο ή ίσο του 10, έχοντας αρχικοποιηθεί στο 1
- Κάθε φορά εκτυπώνει ένα οριζόντιο (χωρίς αλλαγή γραμμής) αστεράκι

Αστεράκια (2) – count up


```java
package gr.aueb.cf.ch3;

/**
 * Εμφανίζει 10 οριζόντια αστεράκια,
 * δηλαδή 1 αστεράκι 10 φορές.
 */
public class Stars10App {

    public static void main(String[] args) {

        int i = 1;

        while (i <= 10) {
            System.out.print("*");
            i++;
        }
    }
}
```

- Είναι σημαντικό να καταλάβει κανείς πόσες φορές εκτελείται η επανάληψη
- Αυτό έχει να κάνει: (1) με την **αρχική τιμή** της μεταβλητής ελέγχου, στο παράδειγμα το i έχει αρχική τιμή 1, (2) την **τελική τιμή** του ελέγχου, στο παράδειγμα είναι 10, και (3) το **βήμα αύξησης** μέσα στο σώμα της while, στο παράδειγμα είναι i++, δηλαδή το i αυξάνεται κατά 1 σε κάθε επανάληψη

Αστεράκια (3) – count down


```java
package gr.aueb.cf.ch3;

/**
 * Εκτυπώνει 10 αστεράκια (10 φορές
 * από ένα αστεράκι).
 */
public class Stars10CountDown {

    public static void main(String[] args) {

        int i = 10;

        while (i >= 1) {
            System.out.print("*");
            i--;
        }
    }
}
```

- Το αποτέλεσμα είναι ίδιο όπως πριν αλλά τώρα ξεκινάμε από το 10 και πάμε στο 1 μειώνοντας κατά 1 μέσα σε κάθε iteration

n Αστεράκια


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Εμφανίζει 10 οριζόντια αστεράκια,
 * δηλαδή 1 αστεράκι 10 φορές.
 */
public class Stars10App {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int i = 1;
        int n = 0;

        System.out.println("Insert number of iterations");
        n = in.nextInt();

        while (i <= n) {
            System.out.print("*");
            i++;
        }
    }
}
```

- Όπως πριν, αλλά το πόσες φορές θα εκτελεστεί το while (πόσα αστεράκια θα εμφανίσει) εξαρτάται από τον αριθμό που θα δώσει ο χρήστης, το n
- Επομένως, αρχική τιμή του i είναι 1, η τελική τιμή ελέγχου δίνεται από το n, που είναι η τιμή που θα δώσει ο χρήστης, και το βήμα αύξησης του i είναι 1
- Επομένως θα εκτελεστεί τόσες φορές, όσο η τιμή του n, και θα εκτυπώσει n αστεράκια

Άθροισμα 10 πρώτων ακεραίων


```java
package gr.aueb.cf.ch3;

/**
 * Υπολογίζει το άθροισμα των 10 πρώτων
 * αριθμών, από το 1-10.
 */
public class Sum10 {

    public static void main(String[] args) {
        int sum = 0;
        int i = 1;

        while (i <= 10) {
            sum = sum + i;  // ή sum += i;
            i++;
        }

        System.out.println("Sum = " + sum);
    }
}
```

- Για να υπολογίσουμε το άθροισμα των 10 πρώτων αριθμών, 1-10, χρειαζόμαστε 10 επαναλήψεις όπου σε κάθε επανάληψη κάνουμε το ίδιο πράγμα, προσθέτουμε την πρώτη φορά το 1, μετά το 2, μετά το 3, κλπ. μέχρι και το 10
- Το i, αν ξεκινήσει από το 1 και πάει μέχρι το 10, μας δίνει αυτά τα 1, 2, 3, …, 10 και **οπότε μπορούμε να προσθέτουμε το i στο sum, όπου αρχικά το sum είναι 0**. Το i αυξάνεται κατά 1 μέσα στο σώμα της while
- Η **αρχική τιμή του sum πρέπει να είναι μηδέν (0)** για να υπολογιστεί σωστά το άθροισμα

Άθροισμα n πρώτων ακεραίων


```java
import java.util.Scanner;

/**
 * Υπολογίζει το άθροισμα των 10 πρώτων
 * αριθμών, από το 1-10.
 */
public class SumNApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int sum = 0;
        int i = 1;
        int n = 0;

        System.out.println("Please insert n");
        n = in.nextInt();

        while (i <= n) {
            sum = sum + i;  // ή sum += i;
            i++;
        }

        System.out.println("Sum = " + sum);
    }
}
```

- Όπως πριν, αλλά η while θα εκτελεστεί n φορές, όπου n είναι ο ακέραιος που δίνει ο χρήστης
- Επομένως σε κάθε επανάληψη, θα προσθέτουμε στο sum, το i και τελικά θα υπολογίσουμε το 1+2+3+…+n
- Το sum = sum + i; μπορεί να γραφεί πιο σύντομα sum += i;

Γινόμενο 10 πρώτων


```java
package gr.aueb.cf.ch3;

/**
 * Υπολογίζει το γινόμενο των 10 πρώτων
 * ακεραίων.
 */
public class Mul10 {

    public static void main(String[] args) {
        // το 1 είναι το ουδέτερο στοιχείο του πολ/μού
        int result = 1;
        int i = 1;

        while (i <= 10) {
            result = result * i;    // ή result *= i;
            i++;
        }

        System.out.println("Result = " + result);
    }
}
```

- Όπως στην πρόσθεση, έτσι και στον πολλαπλασιασμό των 10 πρώτων ακεραίων, το while εκτελείται 10 φορές και κάθε φορά πολλαπλασιάζει το result * i (το result την 1η φορά πρέπει να είναι 1 για να γίνει σωστά ο πολλαπλασιασμός)
- Επομένως, υπολογίζουμε το 1*2*3*…*10

Γινόμενο n πρώτων αριθμών


```java
/**
 * Υπολογίζει το γινόμενο των n πρώτων
 * ακεραίων.
 */
public class MulGeneric {

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

        // το 1 είναι το ουδέτερο στοιχείο του πολ/μού
        int result = 1;
        int n = 0;
        int i = 1;

        System.out.println("Please insert n");
        n = in.nextInt();

        while (i <= n) {
            result = result * i;    // ή result *= i;
            i++;
        }

        System.out.println("Result = " + result);
    }
}
```

- Όπως πριν, μόνο που τώρα η τελική τιμή ελέγχου δίνεται από τον χρήστη, οπότε στη while έχουμε i <= n
- Παρατηρούμε επίσης ότι αν ο χρήστης δώσει 0, τότε η while δεν θα εκτελεστεί καμία φορά, αφού i < 0 θα είναι false (αφού το i έχει αρχική τιμή 1)
- Οπότε τότε το result θα είναι 1 (η αρχική τιμή του result είναι 1), που μπορούμε εξορισμού να υποθέσουμε ότι είναι σωστή, ότι δηλαδή αν n είναι 0, τότε το result είναι 1

Generic while


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Ο χρήστης δίνει αρχική τιμή, τελική τιμή
 * και step και το πρόγραμμα υπολογίζει πόσες
 * φορές εκτελείται μία while.
 */
public class GenericWhileApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int start = 0;    0;
        int endValue = 0;
        int step = 0;
        int counter = 0;

        System.out.println("Please insert, beginValue, EndValue and Step");
        start = in.nextInt();
        endValue = in.nextInt();
        step = in.nextInt();

        while (start <= endValue) {
            counter += 1;
            start += step;
        }

        System.out.println("Iterations count: " + counter);
    }
}
```

```
Run:  GenericWhileApp
"C:\Program Files\Amazon Corretto\jdk11.0.10_
Please insert, beginValue, EndValue and Step
1 10 3
Iterations count: 4
```

	beginValue	endValue	step	count
1	1	10	3	1
2	4	10	3	2
3	7	10	3	3
4	**10**	**10**	**3**	**4**
5	13	10		

- Εκτελείται 4 φορές. Στον πίνακα εμφανίζονται οι τιμές των μεταβλητών
- Την 5η φορά δεν μπαίνει στο while γιατί το beginValue <= endValue είναι false

Nested While


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Nested while.To Generic While τρέχει
 * τρεις φορές.
 */
public class GenericWhileApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int beginValue = 0;
        int endValue = 0;
        int step = 0;
        int counter = 0;
        int times = 1;

        while (times <= 3) {
            System.out.println("Please insert, beginValue, EndValue and Step");
            start = in.nextInt();
            endValue = in.nextInt();
            step = in.nextInt();

            while (start <= endValue) {
                counter += 1;
                start += step;
            }

            System.out.println("Iterations count: " + counter);
            times++;
        }
    }
}
```

- Παρατηρούμε το εξωτερικό while που ελέγχει πόσες φορές θα εκτελεστεί το εσωτερικό while
- Η μεταβλητή *times* ελέγχει το εξωτερικό while το οποίο εκτελείται 3 φορές
- Το εσωτερικό while είναι το GenericApp, του προηγούμενου παραδείγματος

Πλήθος Θετικών Αριθμών


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Μετράει το πλήθος των θετικών
 * αριθμών που δίνει ο χρήστης.
 */
public class PositivesCountApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num;
        int positivesCount = 0;

        System.out.println("Please give a num (int)");
        num = in.nextInt();
        while (num >= 0) {
            positivesCount++;
            System.out.println("Please give a num (int)");
            num = in.nextInt();
        }
        System.out.println("Positive-count: " + positivesCount);
    }
}
```

- Διαβάζουμε αριθμούς όσο υπάρχει θετικός ή μηδέν. Εδώ η μεταβλητή ελέγχου είναι ο αριθμός που διαβάζουμε
- Διαβάσουμε μέχρι να βρούμε αρνητικό (**sentinel value**) σταματάει η while
- Στο σώμα της while γίνεται update η τιμή της num, από τον χρήστη που δίνει ακέραιους αριθμούς (με int.nextInt())
- Επίσης μέσα στο σώμα της while αυξάνουμε κατά 1 την count, κάθε φορά που διαβάζει θετικό
- Παρατηρούμε επίσης πως αφού num >= 0 δεχόμαστε και το 0 ως θετικό αριθμό και το προσμετράμε

Sentinel Values


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Keeps reading until a sentinel
 * value is provided.
 */
public class SentinelApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num = 0;
        int iterations = 0;

        System.out.println("Please provide a num (-1 for Quit)");
        num = in.nextInt();
        while (num != -1) {
            iterations++;
            System.out.println("Please provide a num");
            num = in.nextInt();
        }
        System.out.printf("%d iterations", iterations);
    }
}
```

- Η λογική της while είναι λογική **sentinel values** και όχι γνωστού από πριν αριθμού επαναλήψεων
- Κάνουμε επαναληπτικά κάτι μέχρι να **βρούμε μία ειδική τιμή, μία τιμή 'φρουρό' (sentinel)** η οποία σταματάει το iteration
- Αν για παράδειγμα διαβάζαμε ακεραίους μέχρι να βρούμε το -1 τότε θα είχαμε το πρόγραμμα αριστερά

Ύψωση σε δύναμη


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Υπολογίζει το a^b.
 */
public class PowerApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int a = 0;
        int b = 0;
        int result = 1;
        int i = 1;

        System.out.println("Please insert a, b");
        a = in.nextInt();
        b = in.nextInt();

        while (i <= b) {
            result = result * a;
            i++;
        }
        System.out.printf("%d^%d = %d", a, b, result);
    }
}
```

- Θέλουμε να υπολογίσουμε το a εις την b. Αυτό σημαίνει πως το αποτέλεσμα το βρίσκουμε αν πολλαπλασιάσουμε b φορές το a, δηλαδή a*a* .. a, b φορές. Αυτό σημαίνει μία while που θα εκτελείται b φορές (από i =1 έως b) και κάθε φορά θα πολλαπλασιάζει επί a
- Αρχικοποιούμε το result στο 1 μιας και έχουμε πολλαπλασιασμό
- Το while τρέχει όσο το i <= b δηλ. b φορές αφού αρχικά το i = 1 και μέσα στο loop γίνεται i++. Σε κάθε loop επίσης πολλαπλασιάζουμε το αποτέλεσμα επί το a.
- Αν δώσουμε για παράδειγμα 2^16 θα τρέξει το while 16 φορές και κάθε φορά θα πολλαπλασιάζει επί 2, που είναι αυτό που θέλουμε
- Στην περίπτωση επίσης, που το b είναι 0, δεν θα μπει στο while και τότε το result θα παραμείνει 1 που είναι σωστό μιας και a^0 είναι 1

Συνθήκη επανάληψης (1)

- ***Αν a = 2 και b = 16***, τότε αρχικά i=1 και b=16, οπότε η συνθήκη while (i <= b) είναι αληθής και μπαίνουμε στη while
- Το result, που μέσα στη while γίνεται result * a, την 1η φορά γίνεται ***1*2 δηλ. 2***, ενώ το i, που μέσα στη while γίνεται ***i++*** την 1η φορά γίνεται 2, αφού αρχικά ήταν 1

Συνθήκη επανάληψης (2)

- Στη συνέχεια (στο 2ο loop) ξανά-υπολογίζεται η συνθήκη while (i <= b) να είναι 2 <= 16, οπότε είναι true και το result γίνεται 2*2 = 4, ενώ το i++ γίνεται 3
- ***Γενικά το i αυξάνεται κατά 1 μέσα στο σώμα του while***, όσο το while είναι true, οπότε ***το while θα εκτελεστεί 16 φορές*** για i=1, 2, 3, 4, …, 16
- Όταν το i γίνει 17 τότε η συνθήκη i <= b γίνεται ψευδής και βγαίνουμε από το loop
- ***Τότε το result θα είναι 65536***, που είναι το αναμενόμενο αποτέλεσμα για το 2^16

Παραγοντικό του n (1)


```java
/**
 * Υπολογίζει το n! (n παραγοντικό),
 * δηλαδή 1*2*3*..*n. Εξ ορισμού είναι
 * 0! = 1. Είναι παρόμοιο με το γινόμενο
 * των n πρώτων αριθμών.
 */
public class FactorialApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n = 0;
        int facto = 1;
        int i = 1;

        System.out.println("Please insert n");
        n = in.nextInt();

        while (i <= n) {
            facto = facto * i;  // facto *= i;
            i++;
        }

        System.out.printf("%d! = %d", n, facto);
    }
}
```

- Θέλουμε να υπολογίσουμε το παραγοντικό του n, δηλαδή 1*2*3*.. *n. Αυτό σημαίνει πως χρειαζόμαστε n επαναλήψεις. Κάθε φορά πολλαπλασιάζουμε επί i
- Το facto αρχικοποιείται στο 1, ως ουδέτερο στοιχείο του πολλαπλασιασμού
- Το i αρχικοποιείται στο 1, ως αρχική τιμή της μεταβλητής ελέγχου
- Το while τρέχει από 1 έως n (i <= n) και το βήμα αύξησης είναι i++, οπότε η while εκτελείται n φορές και κάθε φορά πολλαπλασιάζει το αποτέλεσμα επί το i το οποίο αυξάνεται κάθε φορά κατά 1 (δηλ. 1*2*3*..*n). Αυτό είναι το παραγοντικό του n

Παραγοντικό του n (2)

- Αρχικά i = 1 και ανάλογα τι αριθμό θα δώσει ο χρήστης, η συνθήκη while θα είναι αληθής ή ψευδής.
- Αν ο χρήστης δώσει 0 τότε η συνθήκη while είναι ψευδής και δεν θα μπούμε μέσα. Το αποτέλεσμα ωστόσο θα είναι 1 γιατί το facto έχει αρχικοποιηθεί στο 1. Αυτό είναι σωστό γιατί 0! = 1 εξορισμού. Τις ακραίες (οριακές) συνθήκες πρέπει πάντα να τις εξετάζουμε ώστε να λειτουργούν σωστά.
- Αν ο χρήστης δώσει αριθμό μεγαλύτερο από 0, τότε θα μπει στη while και η while θα εκτελεστεί τόσες φορές όσο ο αριθμός που έδωσε ο χρήστης
- Μέσα στη while σε κάθε loop πολλαπλασιάζουμε με i και αυξάνουμε το i κατά 1

```
while (i <= n) {
    facto = facto * i;
    i++;
}
```

Ειδικές Μορφές της While


```java
package gr.aueb.cf.ch3;

/**
 * Infinite while loop.
 */
public class EternalWhile {

    public static void main(String[] args) {
        int i = 1;

        while (i < 0) {
            System.out.println("NEVER gets in");
        }

        while (i <= 1) {
            System.out.println("Only ONE iteration");
            i++;
        }

        while (true) {
            System.out.println("I am ETERNAL!");
        }
    }
}
```

- Η εντολή while **μπορεί να μην εκτελεστεί καμία φορά** αν η συνθήκη ελέγχου είναι false
- **Μία φορά** αν το beginValue είναι ίσο με το endValue και υπάρχει τελεστής ισότητας
- Επίσης, αν η while είναι πάντα true, η while **θα εκτελείται ανένδως**

Εντολή do … while

- Η διαφορά while-do και do-while είναι πως **η do-while εκτελείται οπωσδήποτε μια φορά.** Συντάσσεται ως εξής:
- ```
  do {
      εντολή;
      εντολή;
      …..
      update την παράσταση αλήθειας
   } while <συνθήκη>
  ```
- Πρώτα μπαίνουμε στο σώμα της do … while **εκτελούνται οι εντολές οπωσδήποτε μία φορά** και μετά αποτιμάται η συνθήκη εισόδου (και ταυτόχρονα εξόδου).
- Αν είναι αληθής (true) συνεχίζει μέχρι να γίνει ψευδής (false) οπότε και σταματάει
  ```

Do .. While - Μενού


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Εμφανίζει ένα Μενού, μέχρι ο χρήστης
 * να επιλέξει exit.
 */
public class DoMenuApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int choice = 0;

        do {
            System.out.println("Επιλέξτε ένα από τα παρακάτω:");
            System.out.println("1. Εισαγωγή Προϊόντος");
            System.out.println("2. Διαγραφή Προϊόντος");
            System.out.println("3. Έξοδος");
            choice = in.nextInt();
        } while (choice != 3);

        System.out.println("Thanks for using our app");
    }
}
```

- Τα Μενού θέλουμε να εκτελούνται/εμφανίζονται οπωσδήποτε μια φορά, επομένως είναι ένα καλό παράδειγμα της do-while
- Αφού μπει σίγουρα μία φορά, στη συνέχεια ελέγχεται η choice, η οποία γίνεται update μέσα στην do-while
- Μόλις η choice γίνει 3, η συνθήκη ελέγχου γίνεται false και βγαίνουμε από την do-while

Πλήθος ψηφίων ακεραίου


```java
/**
 * Υπολογίζει το πλήθος των ψηφίων ενός ακεραίου,
 * διαιρώντας επαναληπτικά με 10 μέχρι το αποτέλεσμα
 * της διαίρεσης να γίνει 0.
 */
public class DigitCount {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int inputNum = 0;
        int num = 0;
        int count = 0;

        System.out.println("Please insert a number (int)");
        inputNum = in.nextInt();

        num = inputNum;
        do {
            count++;
            num = num / 10;
        } while (num != 0);

        System.out.printf("Num: %d consists of %d digits", inputNum, count);
    }
}
```

Με do … while διαιρούμε το *num* διαδοχικά με το 10 (όχι το *inputNum* για να μην χάσουμε τον αρχικό αριθμό) μέχρι η διαίρεση να δώσει 0. Αυξάνουμε το count σε κάθε iteration

- Θέλουμε να υπολογίσουμε το πλήθος των ψηφίων ενός ακεραίου, για παράδειγμα του **155, που έχει 3 ψηφία**
- Γνωρίζοντας ήδη από το προηγούμενο κεφάλαιο τη λειτουργία του mod (%) και div (/) ξέρουμε πως κάθε φορά που διαιρούμε ένα ακέραιο με το 10, το αποτέλεσμα είναι τα αριστερά ψηφία και αφήνουμε δεξιά ένα ψηφίο Δηλ. 155 / 10 = 15 και μένει το 5 δεξιά ως υπόλοιπο.
- Αυτό το 5 είναι ένα ψηφίο. Αν διαιρέσουμε το 15 / 10 = 1 μένει το άλλο 5 δεξιά. Αυτό είναι το 2ο ψηφίο
- Αν διαιρέσουμε πάλι 1 / 10 = 0 μένει το 1. Αυτό είναι το 3ο ψηφίο και ο αλγόριθμος τελειώνει αφού το αποτέλεσμα της διαίρεσης είναι 0

Αλγόριθμος

- Χρησιμοποιούμε ***do .. while*** γιατί ο αριθμός θα έχει τουλάχιστον 1 ψηφίο, οπότε θα μπούμε οπωσδήποτε μία φορά στο while
- Η do-while που θα εκτελεστεί τόσες φορές όσο το αποτέλεσμα της διαίρεσης είναι διάφορο από το 0

Break και continue


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Δείχνει τη χρήση των break και
 * continue.
 */
public class BreakContinueApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num = 0;
        int sum = 0;

        while (true) {
            System.out.println("Please provide a num");
            num = in.nextInt();

            // If the number is 5, ignore it
            // and immediately continue to next loop.
            if (num == 5) {
                continue;
            }
            sum += num;

            // If the number equals 10 then
            // stop immediately the loop
            // and get out of while.
            if (num == 10) {
                break;
            }
        }

        System.out.println("Sum: " + sum);
    }
}
```

- Η εντολή ***break*** διακόπτει το loop ενώ η ***continue*** συνεχίζει στο επόμενο loop (χωρίς να εκτελεστούν οι επόμενες εντολές)

Η δομή ελέγχου if (1)

- Η δομή ελέγχου ***if-then-else*** είναι **διακλάδωση**. Συντάσσεται ως εξής (δεν περιλαμβάνεται το then, απλά υπονοείται):
- If (παράσταση) { εντολή/ες }
- If (παράσταση) { εντολή/ές } else { εντολή/ές }

Η δομή ελέγχου if (2)

- Αν στο σώμα της if περιέχεται μία μόνο εντολή δεν χρειάζονται { } αλλά καλό είναι να τα βάζουμε και για μία μόνο εντολή (για να αποφύγουμε κάποια ειδικά προβλήματα που μπορεί να δημιουργηθούν)
- Στην εντολή if-then-else αποτιμάται η παράσταση. Αν είναι true εκτελείται η εντολή/ες του if, αλλιώς εκτελούνται η εντολή/ές του else
- Το else δεν είναι απαραίτητο να υπάρχει

Απλοποιήσεις

- x == true ισοδύναμο με σκέτο x
  - Αντί δηλαδή να πούμε if (x == true) μπορούμε απλά να πούμε: if (x)
- x != true ισοδύναμο με !x
  - Αντί δηλ. να πούμε if (x != true) οπότε τότε σημαίνει πως το x είναι false, μπορούμε απλά να πούμε: if (!x)

Bingo App


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

public class BingoIfApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int key = 10;
        int inputNum = 0;

        System.out.println("Please guess the key number");
        inputNum = in.nextInt();

        if (inputNum == key) {
            System.out.println("Bingo!");
        }
    }
}
```

- Αν ο χρήστης μαντέψει σωστά τον αριθμό-κλειδί, εμφανίζεται το Bingo
- Καλό θα ήταν ωστόσο να καλυφθεί και η περίπτωση που δεν μαντέψει σωστά
- Ας δούμε την επόμενη διαφάνεια

Bingo 2 App


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

public class Bingo2App {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int key = 10;
        int inputNum = 0;

        System.out.println("Please guess the key number");
        inputNum = in.nextInt();

        if (inputNum == key) {
            System.out.println("Bingo!");
        } else {
            System.out.println("Try Again");
        }
    }
}
```

- Έχουμε προσθέσει το else για έλεγχο και της περίπτωσης που δεν βρεθεί το key
- Πάντα θα πρέπει να καλύπτουμε όλες τις περιπτώσεις σε δομές διακλάδωσης όπως η if, διαφορετικά ο κώδικας θα αποτύχει να αντιμετωπίσει όλες τις πιθανές περιπτώσεις εισόδου δεδομένων, όπως θα έπρεπε ώστε ο κώδικάς μας να είναι αξιόπιστος

Bingo with do-while


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Guesses repeatedly until the key will eventually be
 * found. This method of repeated guesses in
 * cryptography, for guessing passwords or cryptographic
 * keys, is called brute-force.
 */
public class BingoWhileApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int key = 10;
        int inputNum = 0;
        boolean bingo = false;

        do {
            System.out.println("Please guess the key number");
            inputNum = in.nextInt();

            if (inputNum == key) {
                bingo = true;
                System.out.println("Bingo! Great guess!");
            } else {
                System.out.println("Try Again");
            }
        } while (!bingo);
    }
}
```

- Το if βρίσκεται μέσα σε μία do-while που εκτελείται επαναληπτικά, όσο δεν έχει γίνει bingo
- Το bingo είναι μία boolean μεταβλητή που λειτουργεί σαν Flag (σημαία). Έχει αρχικοποιηθεί στο false, και μόλις βρεθεί ο αριθμός, μέσα στην if το bingo γίνεται true
- Οπότε, αμέσως μετά η do-while σταματάει (ends)

Bingo with while (true) & break


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Κάνει Bingo όταν ο χρήστης
 * μαντέψει ένα Secret Key.
 */
public class BingoApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num = 0;
        final int SECRET = 11;

        while (true) {
            System.out.println("Please provide a secret num");
            num = in.nextInt();

            if (num == SECRET) {
                System.out.println("Bingo");
                break;
            } else {
                System.out.println("Try again!");
            }
        }
    }
}
```

1. Χρησιμοποιούμε while με break
2. Το secret είναι final

- Παρατηρούμε ωστόσο ότι ο κώδικας στο if κάνει break, επομένως ακόμα και αν δεν έχουμε else δεν προχωράει παρακάτω
- Μπορούμε επομένως να κάνουμε linearize τον κώδικά μας ώστε να μειώσουμε την πολυπλοκότητα του και να γίνεται ευκολότερα test και debug (βλ. επόμενη διαφάνεια)

Advanced Bingo version (2)


```
6       * Κάνει Bingo όταν ο χρήστης
7       * μαντέψει ένα Secret Key.
8       */
9   public class BingoApp {
10
11      public static void main(String[] args) {
12          Scanner in = new Scanner(System.in);
13          int num = 0;
14          final int SECRET = 11;
15
16          while (true) {
17              System.out.println("Please provide a secret num");
18              num = in.nextInt();
19
20              if (num == SECRET) {
21                  System.out.println("Bingo");
22                  break;
23              }
24
25              System.out.println("Try again!");
26          }
27      }
28  }
```

- Εφόσον στο if υπάρχει **break** ο κώδικας δεν θα συνεχιστεί παρακάτω (το πρόβλημα θα ήταν να εκτελεστεί, και το if και ο κώδικας παρακάτω, δηλαδή και τα δύο branches του προηγούμενου if-then-else)
- Εδώ θα εκτελεστεί ή το if ή το "Try again" (το γεγονός αυτό το εγγυάται η ύπαρξη του break στην if), επομένως δεν αλλάζει η λογική του if-then-else αλλά ωστόσο ο κώδικάς μας είναι linear

Error Handling & linearization


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Διαιρεί δύο ακεραίους. Ελέγχει αν ο num2 είναι 0
 * οπότε δεν μπορεί να προχωρήσει η διαίρεση.
 */
public class LinearApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int numerator = 0;
        int denominator = 0;
        int result = 0;

        System.out.println("Please insert the numerator");
        numerator = in.nextInt();
        System.out.println("Please insert the denominator");
        denominator = in.nextInt();

        if (denominator == 0) System.exit(1);

        result = numerator / denominator;
        System.out.printf("%d / %d = %d", numerator, denominator, result);
    }
}
```

- Διαιρούμε δύο αριθμούς
- Ο παρονομαστής όμως δεν μπορεί να είναι 0. Αυτό το 'λάθος' καλύτερα να το ελέγξουμε με if ψηλά, στην αρχή του κώδικα.
- Το if έχει ένα System.exit(1) που διακόπτει το πρόγραμμα με συνθήκη λάθους 1 (οτιδήποτε εκτός του μηδενός θεωρείται λάθος)
- Αν ο denominator δεν είναι 0, τότε προχωράμε. Ο κώδικας είναι linear (δεν χρειαζόμαστε else)

Error handling - Non-linear code


```
22          if (!(denominator == 0)) {
23              result = numerator / denominator;
24          } else {
25              System.exit(1);
26          }
27
28          System.out.printf("%d / %d = %d", numerator, denominator, result);
29      }
30  }
```

- Εναλλακτικά θα μπορούσαμε να κάνουμε κάτι σαν το παραπάνω που δεν είναι τόσο efficient όσο το linearization

Πολλαπλές συνθήκες ελέγχου


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Διαιρεί δύο ακεραίους. Ελέγχει αν ο num2 είναι 0
 * οπότε δεν μπορεί να προχωρήσει η διαίρεση. Υποθέτουμε
 * επίσης ότι ούτε ο αριθμητής μπορεί να είναι 0.
 */
public class LinearApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int numerator = 0;
        int denominator = 0;
        int result = 0;

        while (true) {
            System.out.println("Please insert the numerator");
            numerator = in.nextInt();
            System.out.println("Please insert the denominator");
            denominator = in.nextInt();

            if ((numerator == 0)) {
                System.out.println("Numerator can not be zero");
                break;
            }
            if (denominator == 0) {
                System.out.println("Denominator can not be zero");
                break;
            }
            result = numerator / denominator;
            System.out.printf("%d / %d = %d", numerator, denominator, result);
        }
    }
}
```

- Παρατηρούμε ότι ο κώδικας απλοποιείται μιας και τα λάθη ελέγχονται ψηλά και ο κώδικας παραμένει linear
- Εναλλακτικά θα έπρεπε να έχουμε δύο if-then-else

Πολυπλοκότητα ελέγχων


```
18          while (true) {
19              System.out.println("Please insert the numerator");
20              numerator = in.nextInt();
21              System.out.println("Please insert the denominator");
22              denominator = in.nextInt();
23
24              if (!(numerator == 0)) {
25                  if (!(denominator == 0)) {
26                      result = numerator / denominator;
27                  } else {
28                      System.out.println("Denominator can not be zero");
29                      break;
30                  }
31              } else {
32                  System.out.println("Numerator can not be zero");
33                  break;
34              }
35
36              System.out.printf("%d / %d = %d", numerator, denominator, result);
37          }
38      }
39  }
```

- Παρατηρούμε **πόσο πολύπλοκος μπορεί να γίνει ο κώδικας** αν δεν ελέγχουμε πρώτα για λάθη

Valid conditions (1)


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Μόνο δύο αριθμοί κερδίζουν.
 * Οι 5 και ο 12. Το 70 είναι ουδέτερο.
 */
public class WinApp {

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

        while (true) {
            System.out.println("Please give the lucky number");
            num = in.nextInt();

            if (num != 12 && num != 5 && num != 70) {
                System.out.println("No Lucky number given");
                break;
            }

            if (num == 5) {
                System.out.println("Found: " + num);
                break;
            } else if (num == 12) {
                System.out.println("Found: " + num);
                break;
            }
        }
    }
}
```

- Όπως τα λάθη πρέπει να τα ελέγχουμε στην αρχή του προγράμματος και να κάνουμε break, exit, κλπ. έτσι και στις περιπτώσεις επιτυχούς εκτέλεσης, είναι καλύτερα να έχουμε ένα σημείο εξόδου του output, όχι δύο όπως εδώ (γραμμές 26 και 29)
- Επίσης, εδώ δεν έχουμε μήνυμα ότι δεν βρέθηκε ούτε το 5 ούτε το 12
- Βλ. επόμενη διαφάνεια

Valid conditions (2)


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Μόνο δύο αριθμοί κερδίζουν.
 * Οι 5 και ο 12. Το 70 είναι ουδέτερο.
 */
public class WinApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num = 0;
        boolean found = false;

        while (!found) {
            System.out.println("Please give the lucky number");
            num = in.nextInt();
            if (num != 12 && num != 5 && num != 70) {
                System.out.println("No Lucky number given");
                break;
            }

            if (num == 5) {
                found = true;
            } else if (num == 12) {
                found = true;
            }

            System.out.println("No lucky number found");
        }

        if (found) System.out.println("Found: " + num);
        else System.out.println("No lucky number found");
    }
}
```

1. Οι έλεγχοι για errors γίνονται ψηλά. Γίνεται break
2. Οι επιτυχείς τιμές δεν δίνονται σε κάθε περίπτωση αλλά στο τέλος γίνεται ένα println

Min value


```java
/**
 * Υπολογίζει τον μικρότερο
 * δύο ακεραίων.
 */
public class MinApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num1 = 0;
        int num2 = 0;
        int min = 0;

        System.out.println("Please insert two ints");
        num1 = in.nextInt();
        num2 = in.nextInt();

        if (num1 < num2) {
            min = num1;
        } else {
            min = num2;
        }

        System.out.printf("The min value between %d and %d is: %d\n", num1, num2, min);
    }
}
```

- Ελέγχει σε μία if αν το num1 είναι μικρότερο από το num2, οπότε το min γίνεται num1
- Αλλιώς γίνεται num2

Τριαδικός τελεστής


```
5   /**
6    * Υπολογίζει τον μικρότερο
7    * δύο ακεραίων με τη χρήση
8    * τριαδικού τελεστή.
9    */
10  public class TernaryOpApp {
11
12      public static void main(String[] args) {
13          Scanner in = new Scanner(System.in);
14          int num1 = 0;
15          int num2 = 0;
16          int min = 0;
17
18          System.out.println("Please insert two ints");
19          num1 = in.nextInt();
20          num2 = in.nextInt();
21
22          min = (num1 < num2) ? num1 : num2;
23
24          System.out.printf("The min value between %d and %d is: %d\n",
25                  num1, num2, min);
26      }
27  }
```

- Παρατηρούμε ότι στο προηγούμενο παράδειγμα της if το min αναφέρεται δύο φορές στην εκχώρηση, μία στην if και μία στην else
- Μπορούμε να κάνουμε πιο σύντομο τον κώδικα της if με τον τριαδικό τελεστή
- Είναι επομένως σαν την if, και συντάσσεται ως (συνθήκη ελέγχου) **? :** όπου το ? είναι το then και το : είναι το else

Απόλυτο (abs)


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Βρίσκει το απόλυτο ενός ακεραίου.
 * Το απόλυτο ενός αρνητικού είναι ο αντίστοιχος
 * θετικός. Και ενός θετικού, ο ίδιος ο θετικός.
 */
public class AbsApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num1 = 0;
        int abs = 0;

        System.out.println("Please insert a num");
        num1 = in.nextInt();

        abs = (num1 > 0) ? num1 : -num1;

        System.out.printf("Abs of %d is: %d", num1, abs);
    }
}
```

- Το απόλυτο ενός αριθμού είναι πάντα θετικός
- Αν ο αριθμός a είναι αρνητικός, τότε το απόλυτο είναι το –a, αλλιώς αν το a είναι θετικός, τότε το απόλυτο είναι το ίδιο το a
- Αυτό μπορεί να εκφραστεί εύκολα με τριαδικό τελεστή
- Με if ο κώδικας θα ήταν verbose (φλύαρος, όχι τόσο σύντομος)

Τριαδικός τελεστής

- **? :**

Παράδειγμα

```
abs = (a > 0) ? a : -a;
```

- Είναι συντομότερο να γράφουμε το παραπάνω παρά το παρακάτω το οποίο είναι πιο verbose σε σχέση με τον τριαδικό τελεστή:

```
if (a > 0) {
     abs = a;
} else {
     abs = -a;
}
```

Πολλαπλά if-then-else (1)


```
5   /**
6    * Υλοποιεί ένα μενού με πολλαπλά
7    * if-the-else.
8    */
9   public class MenuIfFeedbackApp {
10
11      public static void main(String[] args) {
12          Scanner in = new Scanner(System.in);
13          int choice = 0;
14
15          do {
16              System.out.println("Επιλέξτε ένα από τα παρακάτω ή 5 για έξοδο");
17              System.out.println("1. Εισαγωγή");
18              System.out.println("2. Αναζήτηση");
19              System.out.println("3. Διαγραφή");
20              System.out.println("4. Ενημέρωση");
21              System.out.println("5. Έξοδος");
22              choice = in.nextInt();
```

- Εμφανίζουμε το Μενού, μέσα σε μία do-while. Διαβάζουμε την επιλογή στο choice

Πολλαπλά if-then-else (2)


```
24              if (choice == 1) {
25                  System.out.println("Επιτυχής εισαγωγή");
26              } else if (choice == 2) {
27                  System.out.println("Επιτυχής αναζήτηση");
28              } else if (choice == 3) {
29                  System.out.println("Επιτυχής διαγραφή");
30              } else if (choice == 4) {
31                  System.out.println("Επιτυχής ενημέρωση");
32              } else if (choice == 5){
33                  System.out.println("Επιλέξατε έξοδο");
34              } else {
35                  System.out.println("Λάθος επιλογή");
36              }
37          } while (choice != 5);
38
39          System.out.println("Goodbye");
40      }
41  }
```

- Ελέγχουμε για κάθε περίπτωση του choice
- Συνεχίζει όσο το choice δεν είναι 5

Φωλιασμένα if (Nested if) (1)


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Δέχεται ως είσοδο: 1) Σύνολο βαθμολογίας
 * και 2) πλήθος μαθημάτων, και υπολογίζει
 * τον μέσο όρο. Στη συνέχεια δίνει feedback
 * στον χρήστη: 9-10 Άριστα, 7-8 Καλώς,
 * 5-6 Λίαν καλώς, κάτω από 5 Αποτυχία.
 */
public class NestedIfApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int modulesCount = 0;
        int totalMarks = 0;
        int average = 0;

        System.out.println("Please insert the sum of grades");
        totalMarks = in.nextInt();

        System.out.println("Please insert modules' count");
        modulesCount = in.nextInt();
```

- Δηλώνουμε και αρχικοποιούμε τις μεταβλητές μας
- Αν δεν γνωρίζουμε από την αρχή τις μεταβλητές που χρειαζόμαστε, προχωράμε στη λογική της εφαρμογής και γυρίζουμε πίσω και δηλώνουμε

Φωλιασμένα if (Nested if) (2)


```
23          System.out.println("Please insert modules' count");
24          modulesCount = in.nextInt();
25
26          if (modulesCount == 0) {
27              System.out.println("Modules' count can not be zero");
28              System.exit(1);
29          }
30
31          average = totalMarks / modulesCount;
32          if (average < 0 || average > 10) {
33              System.out.println("Error in input data");
34              System.exit(1);
35          }
36
37          if (average >= 9) {
38              System.out.println("Excellent: " + average);
39          } else if (average >= 7) {
40              System.out.println("Very Good: " + average);
41          } else if (average >= 5) {
42              System.out.println("Good: " + average);
43          } else {
44              System.out.println("Fail: " + average);
45          }
46      }
47  }
```

- Πρώτα ελέγχουμε το *modulesCount* με το οποίο θα διαιρέσουμε, αν είναι μηδέν δίνουμε μήνυμα και κάνουμε exit (αν το *modulesCount* είναι 0, το average δεν μπορεί να υπολογιστεί), αλλιώς προχωράμε και υπολογίσουμε το average
- Πάλι με την ίδια λογική πρώτα ελέγχουμε αν το average έχει error (<0 ή >10)
- Όταν έχουμε εμφωλιασμένα if, θέλει προσοχή η σωστή στοίχιση και η χρήση { } ώστε τα else να εφαρμόζονται στα σωστά if
- Όπως έχουμε ξαναδεί πρώτα ελέγχουμε τα errors και μετά προχωράμε.

Παράδειγμα με τριαδικό τελεστή σε printf


```java
package gr.aueb.cf.ch3;

import java.util.Scanner;

/**
 * Βρίσκει το απόλυτο ενός ακεραίου
 * και το εμφανίζει άμεσα στην printf
 */
public class TernaryPrint {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num1 = 0;

        System.out.println("Please insert a num");
        num1 = in.nextInt();

        System.out.printf("Abs of %d is: %d", num1, (num1 > 0) ? num1 : -num1);
    }
}
```

- Η έξοδος της printf, λαμβάνει ως παράμετρο, εκτός από το num1, έναν τριαδικό τελεστή που διαφοροποιείται ανάλογα με την τιμή του num1

Άσκηση (1) cont.

- Θέλουμε να αναπτύξουμε ένα πρόγραμμα που να **αποφαίνεται αν ένα έτος είναι δίσεκτο ή όχι**
- Δηλαδή να προτρέπει τον χρήστη να δώσει ένα έτος από το πληκτρολόγιο, να διαβάζει με Scanner το έτος (ακέραιος), να κάνει την **επεξεργασία** και να εμφανίζει στην οθόνη αν το έτος αυτό είναι δίσεκτο ή όχι.

Άσκηση 1 (cont.)

- Δίσεκτο είναι ένα έτος αν έχει 366 ημέρες, αντί 365. Πότε όμως ένα έτος έχει 366 ημέρες;
- Αν διαιρείται με το 4 **ΕΚΤΟΣ ΕΆΝ**
  - **Διαιρείται** ακριβώς και **με το 100**
  - Αλλά **όχι με το 400**.
- Άλλος τρόπος να το εκφράσουμε πιο απλά είναι: ένα έτος είναι δίσεκτο αν (διαιρείται ακριβώς με το 4 και όχι με το 100) ή αν (διαιρείται ακριβώς με το 400)

Άσκηση 1 cont.

- Σύμφωνα με τον ορισμό του δίσεκτου έτους αν ένα έτος **διαιρείται με το 4** (και όχι με το 100) τότε είναι δίσεκτο
- Για αυτό τα έτη 1904, 1908, 1964, 2004, 2008, 2012, 2016, 2020 **ΕΊΝΑΙ δίσεκτα**

Άσκηση (1)

- Ωστόσο, τα ακόλουθα έτη 1700, 1800, 1900, 2100, 2200, 2300, 2500, 2600 **ΔΕΝ είναι δίσεκτα**, επειδή διαιρούνται μεν με το 4 **αλλά** διαιρούνται και με το 100 αλλά ΟΧΙ και με το 400
- Τα ακόλουθα έτη: 1600, 2000, 2400 **ΕΙΝΑΙ δίσεκτα**, επειδή διαιρούνται με το 4, διαιρούνται και με το 100 αλλά και με το 400 (ή απλά επειδή διαιρούνται με το 400)

Άσκηση (2)

- Εμφανίστε επαναληπτικά ένα μενού με τις παρακάτω επιλογές, το οποίο να επαναλαμβάνεται μέχρι ο χρήστης να δώσει τον αριθμό 5. Για κάθε επιλογή από 1 – 4 θα πρέπει να εμφανίζεται feedback, για παράδειγμα αν ο χρήστης δώσει 1, θα εμφανίζεται το μήνυμα “Επιλέξατε Εισαγωγή». Θα πρέπει επίσης να ελέγχετε αν ο χρήστης δώσει αριθμό < 0 ή > 5 και να δίνετε κατάλληλο μήνυμα.

1. Εισαγωγή
2. Διαγραφή
3. Ενημέρωση
4. Αναζήτηση
5. Έξοδος