

**Σχεδίαση – Ανάπτυξη Εφαρμογών Πληροφορικής**  
**13 Σεπτεμβρίου 2017**

- **Διάρκεια 2:00'**
- **Να απαντηθούν ΟΛΑ (4) τα θέματα.**

**Θέμα 1<sup>o</sup>**

Να δημιουργηθεί η στατική μέθοδος **numberedCells(int r, int c)** η οποία δημιουργεί και επιστρέφει ένα δισδιάστατο διάνυσμα (**r** γραμμών και **c** στηλών) από ακεραίους. Το στοιχείο σε κάθε θέση είναι ο αύξων αριθμός της θέσης (ζεκινώντας από το μηδέν) εάν η αριθμηση γίνεται ανά στήλες και σε κάθε στήλη από επάνω προς τα κάτω.

**Θέμα 2<sup>o</sup>**

Δίνεται η κλάση **Worker** (εργάτης) η οποία χρησιμοποιείται στην μοντελοποίηση μίας απλής εφαρμογής διαχείρισης ομάδων προσωπικού. Κάθε αντικείμενο της κλάσης Worker υλοποιεί τις μεθόδους:

<b>Worker(String name)</b>	Κατασκευαστής. Θέτει το όνομα κάθε εργάτη.
<b>void setName(String newName)</b>	Θέτει το όνομα του εργάτη.
<b>String getName()</b>	Επιστρέφει το όνομα του εργάτη.
<b>String toString()</b>	Εκτυπώνει τον εργάτη (σε μία γραμμή εξόδου)

Να γραφεί κώδικας για την κλάση **WorkTeam** η οποία υλοποιεί την ομάδα προσωπικού χρησιμοποιώντας διάνυσμα. Η κλάση **WorkTeam** που θα αναπτύξετε έχει τις παρακάτω μεθόδους:

<b>WorkTeam(String teamName, int n)</b>	Κατασκευαστής. Δημιουργεί την ομάδα προσωπικού με το συγκεκριμένο όνομα και το πολύ <b>n</b> εργάτες.
<b>void insert(Worker w)</b>	Εισάγει τον εργάτη <b>w</b> στην ομάδα.
<b>void printTeamWorkers()</b>	Τυπώνει τους εργάτες της ομάδας.
<b>int noOfWorkers()</b>	Επιστρέφει τον αριθμό των εργατών της ομάδας.

**Θέμα 3<sup>o</sup> (Κληρονομικότητα)**

Υποθέστε ότι χρειάζεται να υλοποιήσουμε ένα πρόγραμμα που διαχειρίζεται ορθογώνια παραλληλεπίπεδα και κυλίνδρους. Το πρόγραμμα εκτελεί βασικές πράξεις για κάθε ένα από τα στερεά αυτά. Οι πράξεις αυτές περιλαμβάνουν την τοποθέτησή τους στο χώρο και την μετακίνησή τους. Επιπλέον, θέλουμε να μπορούμε να υπολογίσουμε το «εμβαδόν»<sup>1</sup> κάθε στερεού καθώς και τον όγκο του. Τέλος, για κάθε στερεό θέλουμε η μέθοδος **toString** (της κλάσης **Object**) να επιστρέψει ένα περιγραφικό μήνυμα με τα χαρακτηριστικά του κάθε στερεού.

Ορίζουμε τις κλάσεις **Box** και **Cylinder**. Για να ελαχιστοποιήσουμε τον παραγόμενο κώδικα, πρώτα υλοποιούμε την κλάση **Solid** και με βάση αυτή ορίζουμε τις **Box** και **Cylinder** σαν υποκλάσεις της.

Κάθε στερεό έχει:

- **Τη θέση του.** Όλα τα στερεά έχουν έχουν X, Y και Z συντεταγμένες ενός σημείου αναφοράς.
- **Κίνηση.** Όλα τα στερεά μπορούν να κινηθούν. Η κίνησή τους μεταβάλλει τις συντεταγμένες τους.
- **Εμβαδόν και όγκο.**
- **Φραστική Περιγραφή.** Κάθε στερεό αποκρίνεται στο μήνυμα **toString** επιστρέφοντας τα χαρακτηριστικά του γνωρίσματα (τύπος στερεού, θέση, διαστάσεις)

Να υλοποιηθούν η κλάση **Solid** και οι υποκλάσεις της **Box**, **Cylinder**.

**Θέμα 4<sup>o</sup> (Διαπροσωπίες)**

Η κλάση **Item** έχει ως σκοπό την μοντελοποίηση της οντότητας «συσκευασμένο προϊόν». Κάθε συσκευασμένο προϊόν (αντικείμενο που δημιουργήθηκε με βάση την κλάση Item) έχει 3 πεδία. Το όνομα του (τύπου **String**), το βάρος του (τύπου **double**), και την τιμή πώλησης ανά τεμάχιο (τύπου **double**). Κάθε αντικείμενο τύπου Item πρέπει να μπορεί να συγκριθεί με άλλο αντικείμενο του ίδιου τύπου, να υλοποιεί δηλαδή την διαπροσωπία **Comparable**. Η σύγκριση δύο αντικειμένων «συσκευασμένου προϊόντος» γίνεται με βάση το κόστος ανα μονάδα βάρους.

Να γραφεί κώδικας για την κλάση Item η οποία υλοποιεί την διαπροσωπία Comparable και περιλαμβάνει:

- δηλώσεις των πεδίων της κλάσης,
- έναν κατασκευαστή για την κλάση,
- ότι άλλο κρίνετε απαραίτητο ώστε ένα αντικείμενο τύπου Item να μπορεί να συγκριθεί με άλλο ομοειδές αντικείμενο.

Η κλάση Item δεν περιλαμβάνει set μεθόδους για τα πεδία της. Η διαπροσωπία Comparable περιλαμβάνει μόνο την μέθοδο **compareTo()** με την παρακάτω περιγραφή.

<sup>1</sup> Το «εμβαδόν» ενός στερεού ισούται με εμβαδόν της βάσης του. Ο κύλινδρος είναι τοποθετημένος όρθιος.

**int compareTo(T o)**

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

The implementor must ensure  $\text{sgn}(x.\text{compareTo}(y)) == -\text{sgn}(y.\text{compareTo}(x))$  for all  $x$  and  $y$ . (This implies that  $x.\text{compareTo}(y)$  must throw an exception iff  $y.\text{compareTo}(x)$  throws an exception.)

The implementor must also ensure that the relation is transitive:  $(x.\text{compareTo}(y) > 0 \ \&\& \ y.\text{compareTo}(z) > 0)$  implies  $x.\text{compareTo}(z) > 0$ .

Finally, the implementor must ensure that  $x.\text{compareTo}(y) == 0$  implies that  $\text{sgn}(x.\text{compareTo}(z)) == \text{sgn}(y.\text{compareTo}(z))$ , for all  $z$ .

It is strongly recommended, but *not* strictly required that  $(x.\text{compareTo}(y) == 0) == (x.equals(y))$ . Generally speaking, any class that implements the Comparable interface and violates this condition should clearly indicate this fact. The recommended language is "Note: this class has a natural ordering that is inconsistent with equals."

In the foregoing description, the notation  $\text{sgn}(\text{expression})$  designates the mathematical *signum* function, which is defined to return one of -1, 0, or 1 according to whether the value of *expression* is negative, zero or positive.

**Parameters:**

*o* - the object to be compared.

**Returns:**

a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

**Throws:**

NullPointerException - if the specified object is null

ClassCastException - if the specified object's type prevents it from being compared to this object.