Διεπαφές (Interfaces)

Γιώργος Θάνος
gthanos@uth.gr
Γραφείο: Γ5/8, 3ος όροφος
Γκλαβάνη 37

Εισαγωγικά

Κατά την ανάπτυξη προγραμμάτων είναι σημαντικό να είναι τυποποιημένος ο τρόπος αλληλεπίδρασης ανάμεσα σε διαφορετικά συστήματα. Η ύπαρξη ενός "συμβολαίου" το οποίο καθορίζει ακριβώς το πώς μπορεί μία κλάση να αλληλεπιδράσει με μια άλλη διευκολύνει την ανεξάρτητη ανάπτυξη κώδικα πάνω σε προσυμφωνημένες αρχές.

Τα interfaces στη Java λειτουργούν ως προ-συμφωνημένες διεπαφές μεταξύ προγραμματιστών. Κάθε interface τυπικά σχετίζεται με:

1. **τους προγραμματιστές που το χρησιμοποιούν:** οι προγραμματιστές γράφουν την εφαρμογή τους ώστε να χρησιμοποιεί τις μεθόδους του interface.
2. **τους προγραμματιστές που το υλοποιούν:** εφόσον θέλουν να ενσωματώσουν τη λειτουργικότητα που υποδεικνύει το interface σε συγκεκριμένες κλάσεις θα πρέπει να υλοποιήσουν τις μεθόδους του στις κλάσεις αυτές.

Παράδειγμα

Θεωρήστε ένα interface το οποίο ορίζει τη χρήση ενός χρονόμετρου (***Timer***).

Όσοι επιθυμούν να υλοποιήσουν το χρονόμετρο θα πρέπει κατ' ελάχιστο να υλοποιήσουν τις παρακάτω μεθόδους:

- **setTimer:** αρχικοποίηση του timer
- **startTimer:** έναρξη του timer και
- **endTimer:** τερματισμός του timer

Ας υποθέσουμε τώρα ότι ένας κατασκευαστής ηλεκτρικών συσκευών (κλιματιστικά, ψυγεία κ. α.) χρησιμοποιεί Java για τον αυτοματισμό τους και θέλει να χρησιμοποιεί μία κλάση τύπου *Timer* για τις συσκευές που κατασκευάζει προκειμένου να ενσωματώσει τη λειτουργία του χρονόμετρου.

Για να το κάνει αυτό, μπορεί να χρησιμοποιήσει οποιαδήποτε κλάση είναι σύμφωνη με τον παραπάνω *interface Timer*.

Ας υποθέσουμε επίσης ότι υπάρχουν τουλάχιστον δύο διαφορετικές κλάσεις που υλοποιούν το συγκεκριμένο *interface Timer*.

Παράδειγμα

Ας υποθέσουμε επίσης ότι υπάρχουν τουλάχιστον δύο διαφορετικές κλάσεις που υλοποιούν το ***interface Timer***.

Γνωρίζοντας ότι όλες οι πιθανές κλάσεις που θα χρησιμοποιήσει ***θα υλοποιούν υποχρεωτικά*** το συγκεκριμένο interface, αρκεί να σχεδιάσει και να υλοποιήσει το λογισμικό του του ώστε να χρησιμοποιεί το interface αυτό.

Σε αυτή την περίπτωση, σχεδιάζει και γράφει το λογισμικό με γνώμονα τον **interface** και όχι τη συγκεκριμένη κλάση που θα επιλέξει να χρησιμοποιήσει τελικά.

Οποιοσδήποτε προγραμματιστής χρησιμοποιεί το ***interface Timer***, δεν χρειάζεται να γνωρίζει το παραμικρό για την εσωτερική υλοποίηση των κλάσεων που το υλοποιούν.

Επίσης, η επιλογή (προς χρήση) της κλάσης που το υλοποιεί μπορεί να μεταβάλλεται σε διαφορετικές εκδόσεις του προγράμματος, χωρίς να επηρεάζεται ο κώδικας το χρησιμοποιεί interface.

Υιοθετώντας την έννοια του *interface* και γράφοντας κώδικα που χρησιμοποιεί interfaces, γράφουμε κώδικα που είναι ανεξάρτητος από την οποιαδήποτε υλοποίηση των interfaces που χρησιμοποιεί.

Πλεονεκτήματα

Η προτυποποίηση που παρέχει ο μηχανισμός του interface επιτρέπει:

1. την εισαγωγή ενός τυποποιημένου τρόπου αλληλεπίδρασης μεταξύ διαφορετικών συστημάτων του λογισμικού.
2. την εύκολη και χωρίς σφάλματα επαναχρησιμοποίηση κώδικα.
3. την αλλαγή της κλάσης που επιλέγουμε να χρησιμοποιήσουμε ως πάροχο της λειτουργικότητας του interface, χωρίς να επηρεαστεί η λειτουργικότητα του τελικού προγράμματος.
4. την αλλαγή της υφιστάμενης υλοποίησης μιας κλάσης που υλοποιεί ένα interface, δίχως οι χρήστες της να έχουν ενημέρωση για τις εσωτερικές αλλαγές που συντελέστηκαν σε αυτή.

Δήλωση του Interface

```
public interface Person extends Interface1,
 Interface2, Interface3 {


    // constant declarations
    double E = 2.718282;
    double PI = 3.14159;



    // method signatures
    public void interfaceMethod(int i, double x);
    public int interfaceMethod2(String s);
}
```

public ή κανένας προσδιοριστής πρόσβασης: Αν οριστεί ***public*** τότε το interface είναι ορατό από όλες τις κλάσεις και όλα τα πακέτα στην Java. Αν δεν οριστεί κάτι (***package private***) τότε το interface είναι ορατό μόνο μέσα στο πακέτο στο οποίο δηλώνεται.

interface: δεσμευμένη λέξη

το όνομα του interface: στο παράδειγμα μας Person.

extends: ένα ή περισσότερα interfaces τα οποία επεκτείνει το συγκεκριμένο interface (comma seperated).

Στο σώμα του interface ορίζονται

- πεδία τα οποία έχουν σταθερές τιμές (σταθερές).
- μόνο τα prototypes των μεθόδων.

Δήλωση του Interface

Τόσο στις κλάσεις όσο και στα interfaces ορίζεται η ιδιότητα της κληρονομικότητας. Η διαφορά είναι ότι ενώ στην κλάση μπορούμε να έχουμε μόνο μία γονική κλάση, στα interfaces μπορούμε να έχουμε περισσότερα του ενός γονικά interfaces.

Ακριβώς όπως και οι κλάσεις κάθε interface πρέπει να βρίσκεται σε ξεχωριστό αρχείο, όπου το όνομα του αρχείου ταυτίζεται με το όνομα του interface και έχει κατάληξη .java.

Για παράδειγμα το interface με όνομα Person θα βρίσκεται σε ένα αρχείο με όνομα Person.java.

Παράδειγμα interface - Η Στοίβα (Stack)

Interface Stack

Ας υποθέσουμε ότι θέλουμε να δηλώσουμε μέσω ενός interface της μεθόδους που πρέπει να υποστηρίζει μία στοίβα. Η στοίβα είναι μία δομή αποθήκευσης δεδομένων τύπου Last In First Out (LIFO), όπου το αντικείμενο αποθηκεύεται στη στοίβα τελευταίο εξάγεται από αυτή πρώτο. Κάθε στοίβα θα πρέπει να υποστηρίζει κατ' ελάχιστο τα εξής:

- μέθοδο επιστροφής του μεγέθους της στοίβας.
- μέθοδο ένθεσης στην κορυφή της στοίβας.
- μέθοδο απόσβεσης του κορυφαίου στοιχείου.
- μέθοδο επιστροφής του κορυφαίου στοιχείου της στοίβας χωρίς διαγραφή από τη στοίβα.

```
public interface Stack {
  public int size();  //μέγεθος στοίβας
  public void push(Object o); // μέθοδος
                              // ένθεσης
  public Object pop(); // μέθοδος εξαγωγής
  public Object top(); // μέθοδος επιστροφής
    // του 1ου στοιχείου χωρίς διαγραφή.
}
```

Υλοποίηση του interface Stack

Μία κλάση υλοποιεί ένα interface μόνο εάν υλοποιεί ΟΛΕΣ τις μεθόδους του interface. Εάν έστω και μία μέθοδος δεν υλοποιείται τότε η κλάση ΔΕΝ υλοποιεί το interface.

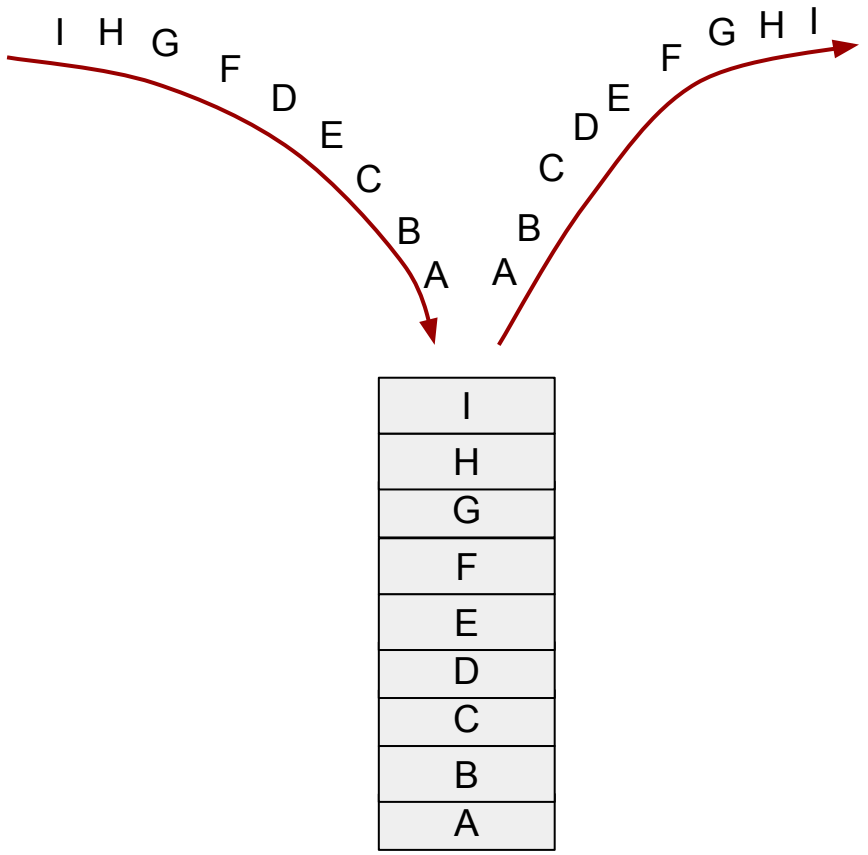
Σε συνέχεια του προηγούμενου παραδείγματος θα επιχειρήσουμε να δημιουργήσουμε δύο διαφορετικές κλάσεις που υλοποιούν το συγκεκριμένο interface. Οι κλάσεις αυτές είναι:

- η **`ArrayStack`** που υλοποιεί τη στοίβα μέσω ενός πίνακα και
- η **`ArrayListStack`** που υλοποιεί τη στοίβα χρησιμοποιώντας την κλάση **`ArrayList`**.

Το interface ως τύπος δεδομένων

Ας υποθέσουμε τώρα ότι θέλουμε να χρησιμοποιήσουμε μία στοίβα για να αντιμεταθέσουμε τα στοιχεία ενός πίνακα χαρακτήρων.

Ο πίνακας αρχικά περιέχει τμήμα της αγγλικής αλφαβήτου (9 χαρακτήρες) . Θέλουμε να αντιστρέψουμε την σειρά με την οποία αποθηκεύονται οι χαρακτήρες στον πίνακα.



Το interface ως τύπος δεδομένων

```
public class ArrayManipulator {
  public static void main(String []args) {
    Character alphabet[] = {'A','B','C','D','E','F','G','H','I'};
    printArray(alphabet); Stack stack = new LinkedStack();
    invertArray(alphabet, stack);  printArray(alphabet);
  }
  public static void invertArray(Object []array, Stack stk) {
    for(int i=0; i<array.length; i++) stk.push(array[i]);
    for(int i=0; i<array.length; i++)array[i] = stk.pop();
  }
  public static void printArray(Character []array) {
    for(Character c : array)
      System.out.print(c+"   "); System.out.println("");
  }
}
```

Παρατηρήστε ότι η μέθοδος invertArray παίρνει ως 2ο όρισμα όχι το αντικείμενο μίας κλάσης αλλά ένα αντικείμενο τύπου **interface Stack**.

Αυτό που δηλώνει η συγκεκριμένη μέθοδος είναι ότι:
1. ως 2ο όρισμα παίρνει ένα αντικείμενο του οποίου η κλάση υλοποιεί το interface Stack.
2. Από τις διαθέσιμες μεθόδους της κλάσης αυτής μπορούν να χρησιμοποιηθούν μόνο οι μέθοδοι που δηλώνονται στο interface Stack.

Σύγκριση Abstract Κλάσεων και Interfaces

Χαρακτηριστικό	Interface	Abstract Class
Πολλαπλή κληρονομικότητα	Μία κλάση μπορεί να κληρονομεί πολλαπλά interfaces	Μία κλάση μπορεί να κληρονομεί μόνο μία abstract κλάση
Default υλοποίηση	Στην Java 8 και μεταγενέστερα ένα interface μπορεί να παρέχει μία default υλοποίηση. Καθώς ένα interface δεν μπορεί να περιέχει πεδία, η πιθανή default υλοποίηση ενός interface θα πρέπει να είναι ανεξάρτητη των πεδίων της κλάσης	Μία abstract κλάση μπορεί να παρέχει default υλοποίηση, η οποία όμως έχει πρόσβαση στα εσωτερικά δεδομένα της κλάσης
Πεδία	Ένα Interface μπορεί να περιέχει μόνο σταθερές δηλ μεταβλητές που είναι static και final	Μία Abstract κλάση μπορεί να περιέχει static, static final και non-static (κανονικές) μεταβλητές
Περιοριστές πρόσβασης	Σε ένα interface όλες οι δηλώσεις θεωρούνται *public*	Μία abstract κλάση μπορεί να έχει επιμέρους προσδιοριστές πρόσβασης για τα πεδία και τις μεθόδους της, όπως *public, protected, package-private, private*.
Λειτουργικότητα	Ένα interface χρησιμοποιείται για να δηλώσει την εξωτερική διεπαφή μίας κλάσης με άλλες κλάσεις	Μία abstract κλάση μπορεί να δηλώσει τόσο της εξωτερική διεπαφή, όσο και λεπτομέρειες της εσωτερικής υλοποίησης
Χρήση	Εάν διαφορετικές υλοποιήσεις κλάσεων χρησιμοποιούν τις ίδιες μεθόδους για την διεπαφή τους με άλλες κλάσεις τότε ένα interface θεωρείται κατάλληλο για να δηλώσει τα κοινά τους χαρακτηριστικά.	Εάν διαφορετικές υλοποιήσεις κλάσεων έχουν κοινά χαρακτιριστικά που αφορούν την εσωτερική τους κατάσταση (π.χ. κοινές μεταβλητές) τότε η χρήση μίας abstract κλάσης ως κοινός πρόγονος είναι η σωστότερη επιλογή.
Ταχύτητα	Λιγότερο ταχύ	Ταχύτερο
Επεκτασιμότητα	Θεωρείται κακή προγραμματιστική πρακτική η επέκταση ενός Interface. Ένα Interface δεν θα πρέπει να σχεδιάζεται με γνώμονα την μελλοντική του επέκταση.	Μία abstract κλάση μπορεί να αλλάξει την υλοποίηση της και να επεκταθεί.

Ο τελεστής *instanceof*

Ο τελεστής ***instanceof*** επιστρέφει boolean true/false εάν ένα αντικείμενο ανήκει σε συγκεκριμένο τύπο δεδομένων. Με χρήση του τελεστή instanceof μπορούμε να εξετάσουμε εάν ένα αντικείμενο αποτελεί υλοποίηση μίας κλάσης ή υλοποιεί ένα συγκεκριμένο interface

Παράδειγμα χρήσης instanceOf

```
public interface Person {}
public class Parent {}
public class Child extends Parent implements Person {}
class InstanceofDemo {
  public static void main(String[] args) {
    Parent obj1 = new Parent(); Parent obj2 = new Child();
    String str = "obj1 instanceof Parent: "+ (obj1 instanceof Parent);
    str += "obj1 instanceof Child: "+ (obj1 instanceof Child);
    str += "obj1 instanceof Person: "+ (obj1 instanceof Person);
    str += "obj2 instanceof Parent: " + (obj2 instanceof Parent);
    str += "obj2 instanceof Child: " + (obj2 instanceof Child);
    str += "obj2 instanceof Person: " + (obj2 instanceof Person);
  }
}
```

```
obj1 instanceof Parent: true
obj1 instanceof Child: false
obj1 instanceof Person: false
obj2 instanceof Parent: true
obj2 instanceof Child: true
obj2 instanceof Person: true
```