Υ325 - Αντικειμενοστρεφής Τεχνολογία

2021

Τμ. ΗΜ&ΤΥ - Πανεπιστήμιο Πατρών

Εβδομάδα 6 - 18/11/2021

Java abstraction

Αφαιρετική κλάση - Abstract class

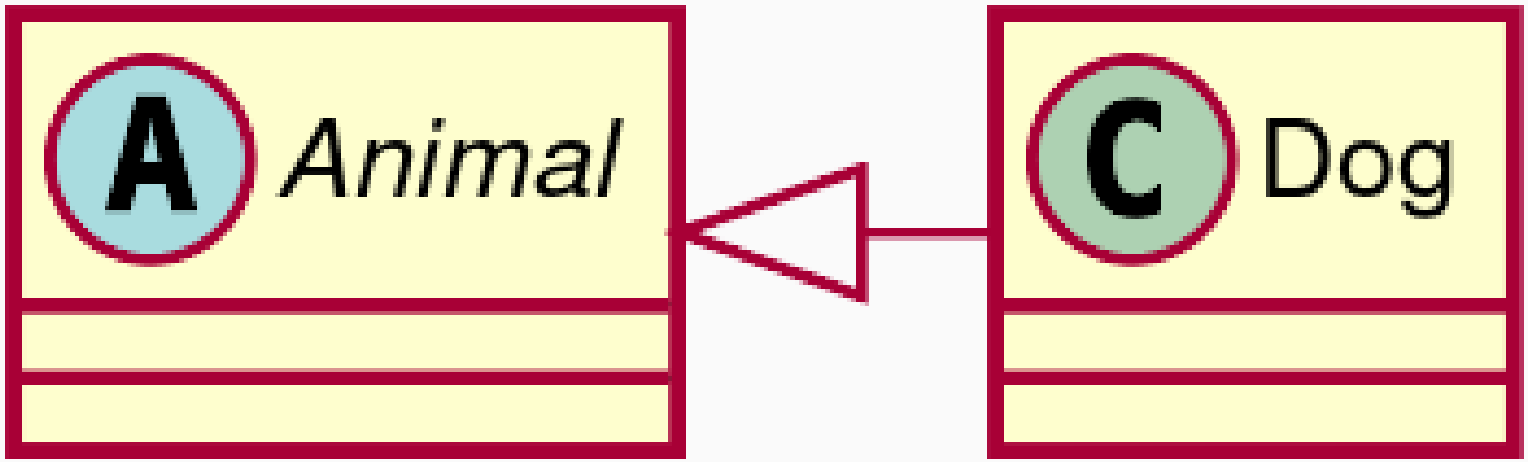
Μια **αφαιρετική** (abstract) κλάση δεν μπορεί να έχει στιγμιότυπο.

Για να χρησιμοποιηθεί πρέπει να την επεκτείνουμε (extend).

```
abstract class Animal {
    void eat() {
        System.out.println("Τρώω...");
    }

    abstract void makeNoise();
}

class Dog extends Animal {
    void makeNoise() {
        System.out.println("Γαβ!");
    }
}
```

Αφαιρετική κλάση - Abstract class (2)

- Δηλώνεται με τη λέξη-κλειδί `abstract`.
- Μπορεί να περιέχει μίξη από `abstract` και μη-`abstract` μεθόδους.
  - μια `abstract` μέθοδος δεν μπορεί να έχει υλοποίηση.
  - είμαστε υποχρεωμένοι να την υλοποιήσουμε όταν κληρονομήσουμε την αφαιρετική κλάση.
- Δεν μπορεί να δημιουργήσει στιγμιότυπα.
- Μπορεί να έχει δημιουργούς και στατικές μεθόδους.
- Μπορεί να έχει μεθόδους που είναι `final`.



Διεπαφή - Interface

Το `interface` είναι ένα *σχέδιο* με οποίο η κλάση πρέπει να συμμορφώνεται.

Μεταβλητές διεπαφής - interface fields

Τυπικά μια διεπαφή *δεν* έχει μεταβλητές, έχει σταθερές.

Όλες οι μεταβλητές που δηλώνονται σε μια διεπαφή είναι `public`, `static` και `final`, αν και δεν χρειάζεται να το δηλώσουμε.

```
    interface Colored {
        //η μεταβλητή είναι στατική - ανήκει στην *κλάση* που υλοποιεί τη
            διεπαφή Colored
        String color = "black";
        //και είναι και final, η δήλωση από πάνω είναι ισοδύναμη με αυτή:
        public static final String color = "black";
    }
```

Διεπαφή - Interface (2)

Το `interface` είναι ένα *σχέδιο* με οποίο η κλάση πρέπει να συμμορφώνεται.

Ένα τέτοιο σχέδιο κλάσεων, το `interface`, μπορεί να περιέχει μεθόδους.

Μέθοδοι διεπαφής - interface methods

Όλες οι μέθοδοι της διεπαφής είναι `public`, ακόμη και αν δεν το γράψουμε.

Μια διεπαφή μπορεί να έχει τρία είδη μεθόδων:

- `abstract`,
- `default`,
- `static`.

Διεπαφή - Interface (3)

Ο πιο συνήθης τύπος μεθόδου διεπαφής είναι η αφαιρετική μέθοδος.

Όλες οι κλάσεις που υλοποιούν το `interface`, πρέπει να υλοποιούν τις μεθόδους του.

```
interface Colored {
    String color = "black";
    //ισοδύναμο με public abstract String getMatchingColor();
    String getMatchingColor();
    //ισοδύναμο με, αλλά συνήθως παραλείπουμε τα public abstract:
    public abstract String getMatchingColor();
}
...
class ComplementaryColorPicker implements Colored {
    public String getMatchingColor() {
        ...
    }
}
```

Interface - default method

Με τη λέξη κλειδί `default` μπορούμε να ορίσουμε μέθοδο που **να έχει υλοποίηση** στο `interface` (από τη Java 8 - 2014):

```
interface Colored {
    default String getRGBValue(String color) {
        ...
        return someRGBValue;
    }
}
```

Οι κλάσεις που υλοποιούν το `interface`, μπορούν να χρησιμοποιήσουν τη `default` μέθοδο ή να την κάνουν override.

Interface - static method

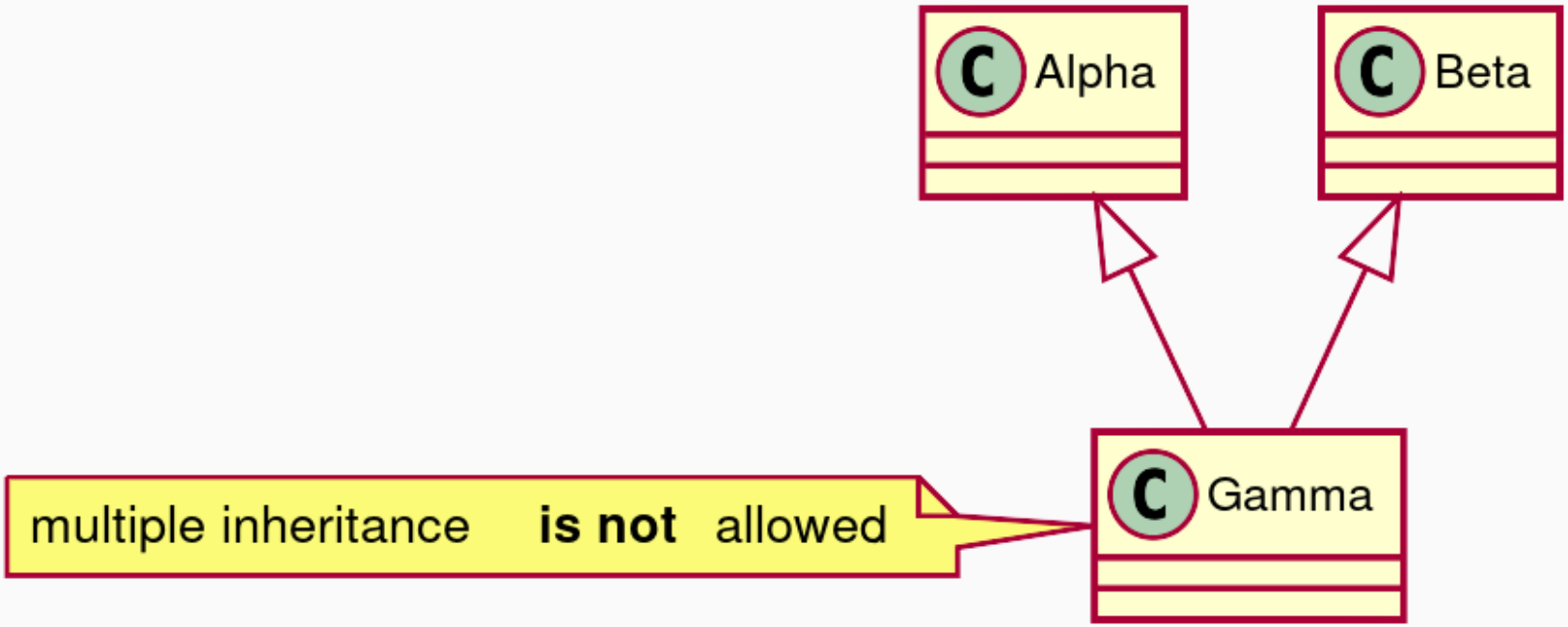
Με τη λέξη κλειδί `static` μπορούμε να ορίσουμε στατική μέθοδο και να ανήκει στον τύπο του `interface` (από τη Java 8 - 2014):

```
interface Colored {
    static String toGrayScale(String color) {
    }
}
...
class Test {
    ...
    Colored.toGrayScale("someColor");
}
```

Πολλαπλή κληρονομικότητα στη Java

Ιστορικά, η Java είχε σχεδιαστεί ώστε να μην επιτρέπει την πολλαπλή κληρονομικότητα:

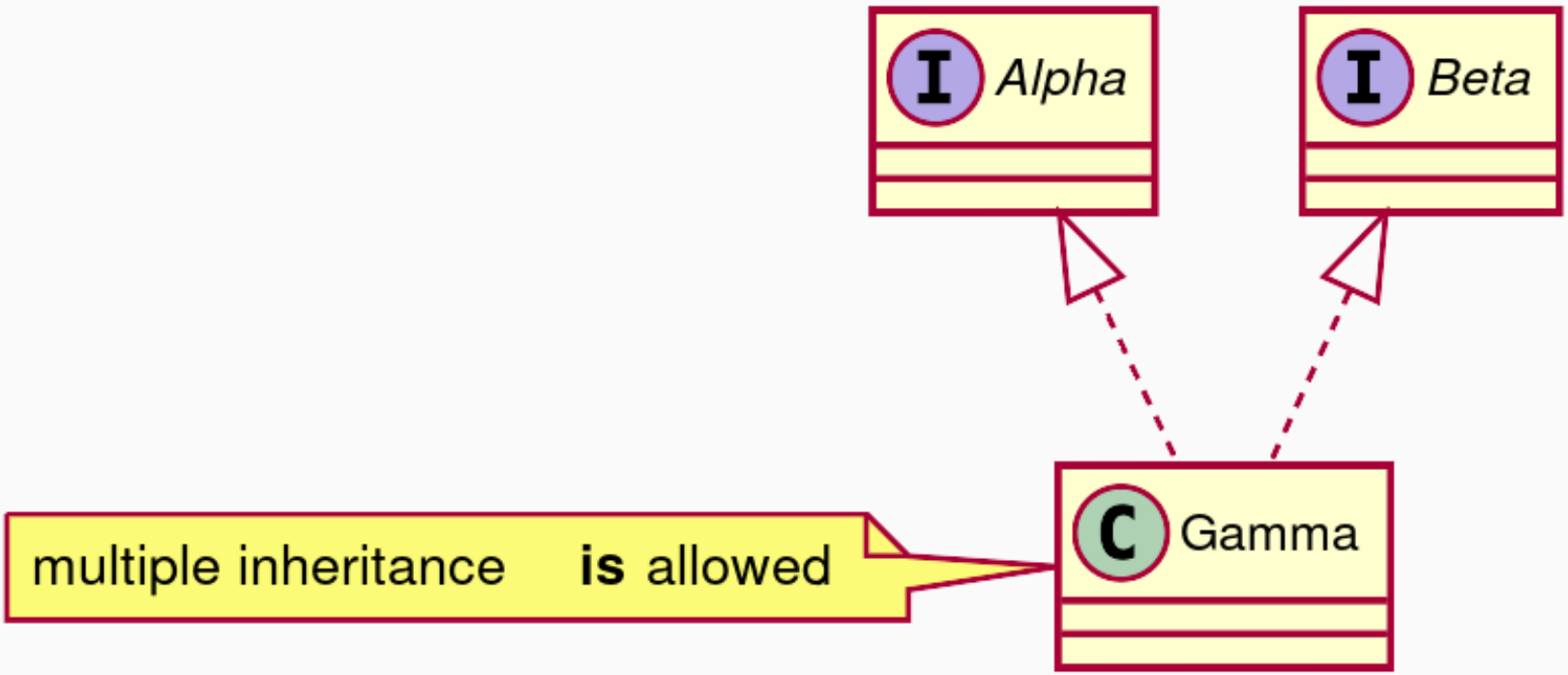
```
class Alpha { void aMethod() {...} }
class Beta { void aMethod() {...} }

//Δεν γίνεται στη Java
class Gamma extends Alpha, Beta {}
```



Πολλαπλή κληρονομικότητα στη Java (2)

Από την Java 8 (2014), επιτρέπονται μέθοδοι `default` στο `interface`.

Με τον τρόπο αυτό μπορούμε να έχουμε πολλαπλή κληρονομικότητα στη Java, αφού επιτρέπεται η υλοποίηση πολλαπλών διεπαφών:



Σημ.: Από την Java 9 (2017) επιτρέπονται και `private` μέθοδοι.

Abstract class Vs. interface

Συνοπτικά, στο `interface`:

- οι μεταβλητές είναι `public static final`,
- οι μέθοδοι είναι `abstract`,
  - εκτός από τις `default` και τις `static`,
- οι μέθοδοι είναι όλες `public`, εκτός από τις `private`,
  - χρησιμοποιούμε τις `private` για να γράψουμε πιο ευανάγνωστα τον κώδικα των `default`,
- το `interface` δεν μπορεί να παράξει στιγμιότυπα, και άρα δεν έχει δημιουργό,
- λέμε πως *υλοποιούμε* ένα `interface`, με τη λέξη-κλειδί `implements`.

Κατά τα άλλα, το `interface` είναι ένας **τύπος** στη Java και μπορεί να χρησιμοποιηθεί παντού όπου δηλώνουμε τύπο.

Abstract class Vs. interface (2)

Μια `abstract` κλάση είναι μια κλάση που δεν μπορεί να έχει στιγμιότυπα.

- Μπορούμε να δηλώσουμε `abstract` μεθόδους, που οι απόγονοι είναι υποχρεωμένοι να υλοποιήσουν,
- δεν υπάρχει άλλος περιορισμός, π.χ. στους access modifiers ή στο είδος των μεταβλητών, σε αντίθεση με τους περιορισμούς του `interface`.

Κοινά στοιχεία είναι:

- δεν μπορούμε να έχουμε στιγμιότυπα ενός `interface` ή μιας `abstract` κλάσης,
- και στα δύο μπορούμε να έχουμε μεθόδους χωρίς σώμα (`abstract` μεθόδους) αλλά και μεθόδους με υλοποίηση.

Abstract class Vs. interface (3)

Χρησιμοποιούμε `interface` αν

- χρειάζεται να αναμείξουμε διαφορετικές ιεραρχίες κλάσεων (δηλ. πολλαπλή κληρονομικότητα),
- χρειάζεται να προδιαγράψουμε μια διεπαφή (τί πρέπει να μπορεί να κάνει μια κλάση) σε κλάσεις που δεν σχετίζονται μεταξύ τους,
  - παραδείγματα από το Java API: `ActionListener`, `Comparable`
- όταν χρειάζεται να προδιαγράψουμε μια διεπαφή σε άλλους προγραμματιστές, χωρίς να μας ενδιαφέρει η υλοποίησή της.

Abstract class Vs. interface (4)

Χρησιμοποιούμε `abstract` κλάση αν

- έχουμε κληρονομικότητα (*"ο τύπος X είναι ένα είδος Y"*) και υπάρχει κοινός κώδικας: τότε μπορούμε να ορίσουμε μια `abstract` κλάση-πρόγονο, που να περιέχει όλον τον κοινό κώδικα,
- προδιαγράφουμε διεπαφή (τί πρέπει να μπορεί να κάνει μια κλάση) και να παρέχουμε ένα μέρος της υλοποίησης,
- χρειαζόμαστε να ορίσουμε πεδία που δεν είναι `public static final`.