# Υ325 - Αντικειμενοστρεφής Τεχνολογία

2021

Τμ. ΗΜ&ΤΥ - Πανεπιστήμιο Πατρών

# Εβδομάδα 5 - 11/11/2021

# Τελεστής instanceof

Είναι ένας δυαδικός τελεστής για να ελέγξουμε αν ένα αντικείμενο είναι κάποιου τύπου.

```
object instanceof Type
```

π.χ.

```
class Round {}

class Circle extends Round {}

public boolean checkInstance() {
    Circle circle = new Circle();
    System.out.println(circle instanceof Round); //true
}
```

# Τελεστής instanceof (2)

Το αποτέλεσμα είναι `true` ή `false`.

```
object instanceof Type
```

Επιστρέφει `true`:

- όταν το `object` είναι στιγμιότυπο του `Type`.
- όταν το `object` είναι στιγμιότυπο μιας υποκλάσης του `Type`.
- όταν το `object` υλοποιεί το interface `Type`.

# Upcasting

```
class Shape {}
class Round extends Shape {}
class Ellipsis extends Round {}
```

# Upcasting (2)

**Upcasting** από την υποκλάση στην υπερκλάση

```
Shape ellipsis = new Ellipsis();
Shape ellipsis = (Shape) new Ellipsis();
```

Η μεταβλητή `ellipsis:Shape` μπορεί να αναφερθεί σε αντικείμενα οποιουδήποτε υποτύπου της `Shape`.

Το upcasting περιορίζει τις μεθόδους που μπορούμε να καλέσουμε: μόνο αυτές της κλάσης `Shape`.

Το upcasted στιγμιότυπο δεν άλλαξε, συνεχίζει να έχει τη συμπεριφορά της `Ellipsis`, αρκεί να γίνει downcasting.

# Downcasting

```
class Shape {}
class Round extends Shape {}
class Ellipsis extends Round {}
```

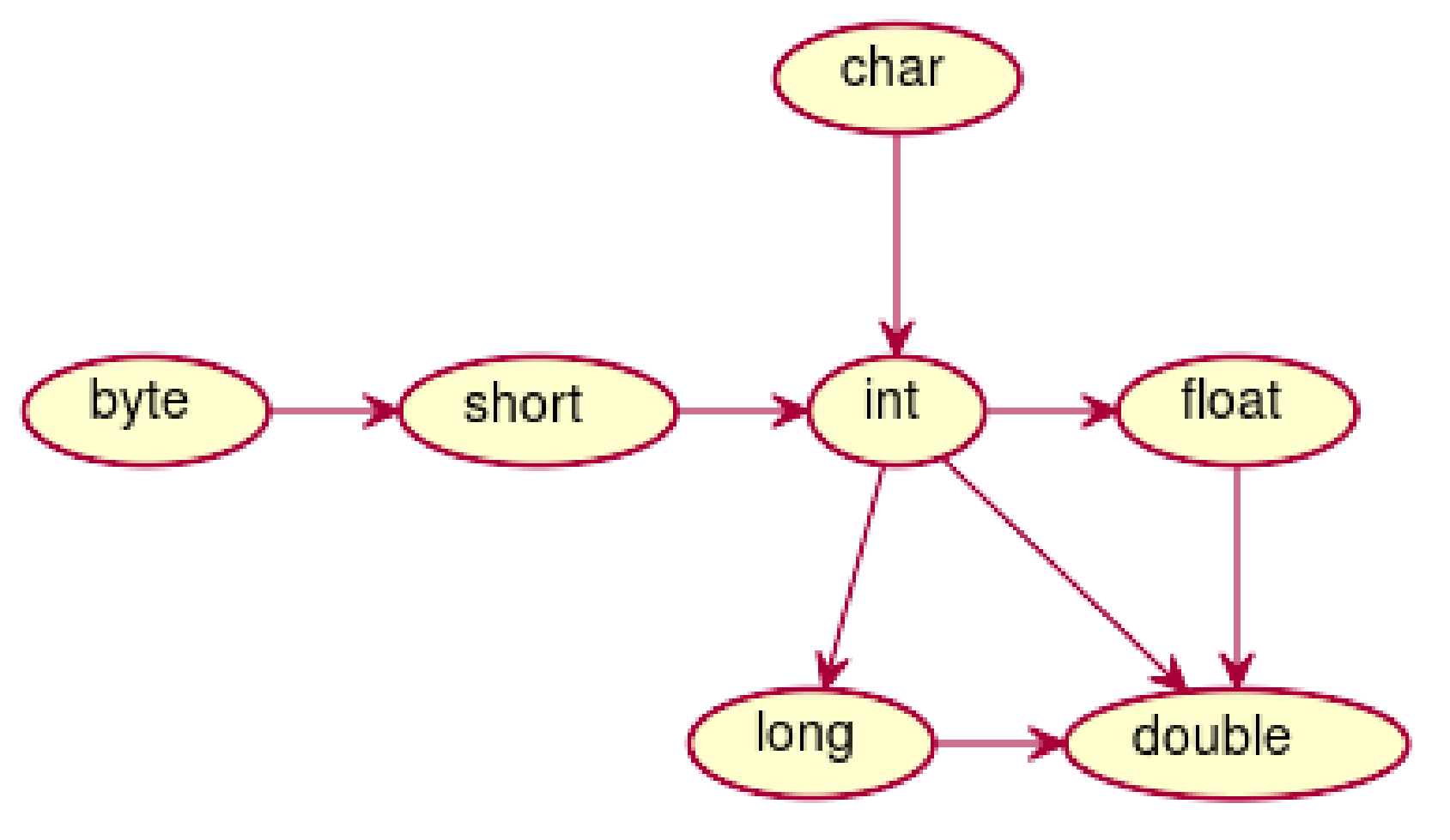
## **Downcasting** από την υπερκλάση στην υποκλάση

```
// σφάλμα μεταγλώττισης
Ellipsis ellipsis = new Shape();

// σφάλμα ClassCastException
Ellipsis ellipsis = (BabyDog) new Shape();

// από Shape->Ellipsis, επιτρέπεται
Shape ellipsis = new Ellipsis();
Ellipsis ellipsis2 = ellipsis;
```

# Type promotion



# Type promotion παράδειγμα

```
class TypePromo {
    void sum(int a, double b) {
        System.out.println(a + b);
    }

    public static void main(String args[]) {
        TypePromo typePromo = new TypePromo();

        //Το δεύτερο όρισμα γίνεται double
        typePromo.sum(20, 20);
    }
}
```

# Java collections

# Java collections

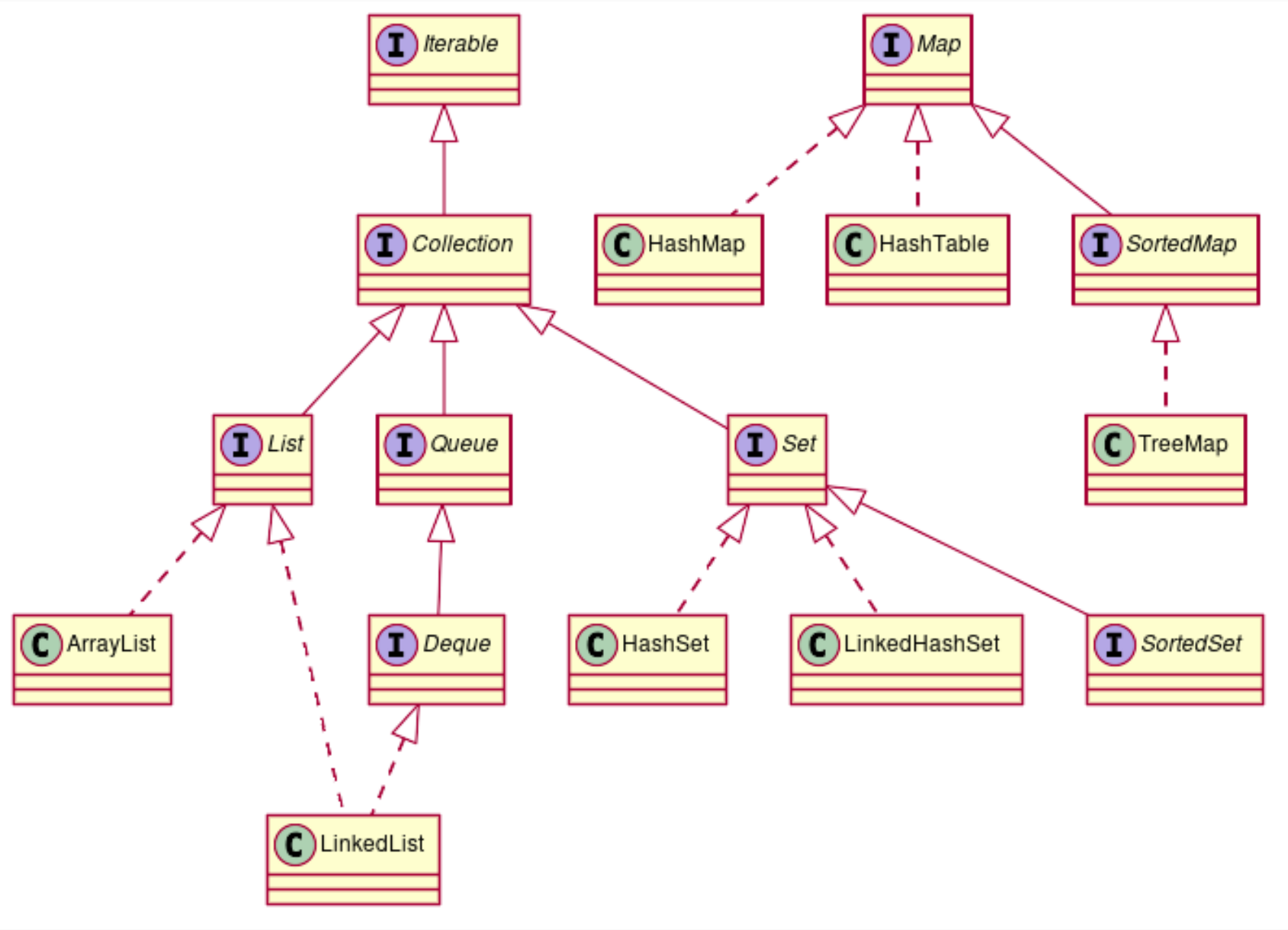
Η Java μας παρέχει έτοιμες δομές δεδομένων, δηλ. συλλογές από οργανωμένα δεδομένα.

Μπορούμε να προσθέσουμε, να αφαιρέσουμε, να αναζητήσουμε, να ανακτήσουμε και να απαριθμήσουμε τα στοιχεία αυτών των δομών.

Το Java Collection Framework περιέχει υλοποιήσεις τέτοιων δομών.

Όλοι οι τύποι του Java Collection Framework κληρονομούν από τα `Iterable` και `Collection`.

# Java Collection Framework (μερική επισκόπηση)



# Μέθοδοι του Collection interface

Μερικές από τις μεθόδους διαθέσιμες στο `Collection` interface.

| Μέθοδος | Λειτουργία |
|---|---|
| `public boolean add(E e)` | Εισαγωγή στοιχείου |
| `public boolean addAll(Collection<? extends E>c)` | Εισαγωγή συλλογής |
| `public boolean remove(Object element)` | Διαγραφή στοιχείου |
| `public boolean removeAll(Collection<?>c)` | Διαγραφή συλλογής |
| `public boolean retainAll(Collection<?>c)` | Διαγραφή όλων εκτός... |
| `public int size()` | Πλήθος στοιχείων |
| `public void clear()` | Διαγραφή όλων |

# Μέθοδοι του Collection interface (2)

Μερικές από τις μεθόδους διαθέσιμες στο `Collection` interface.

| Μέθοδος | Λειτουργία |
|---|---|
| `public boolean contains(Object element)` | Αναζήτηση στοιχείου |
| `public boolean containsAll(Collection<?>c)` | Αναζήτηση συλλογής |
| `public Iterator iterator()` | Επιστρέφει iterator |
| `public Object[] toArray()` | Μετατροπή σε array |
| `public boolean isEmpty()` | Αν είναι κενό |
| `public boolean equals(Object element)` | Συγκρίνει δύο συλλογές |

# Μέθοδοι του Iterator interface

Το `Iterator` επιτρέπει να διατρέξουμε τα στοιχεία προς τα εμπρός.

Μερικές μέθοδοι του `Iterator` interface.

| Μέθοδος | Λειτουργία |
|---|---|
| `public boolean hasNext()` | αν υπάρχει και άλλο στοιχείο |
| `public Object next()` | επιστρέφει το επόμενο στοιχείο |

# Βασικές δομές

- `List`: μια συλλογή όπου η σειρά των στοιχείων έχει σημασία,
- `Set`: δεν περιέχει διπλά στοιχεία,
- `Queue`: first-in, first-out,
- `Map`: τα στοιχεία αποθηκεύονται με *κλειδιά*.

# Υποδιαιρέσεις

| Βασική δομή | Υποδιαίρεση |
|---|---|
| `List` | `ArrayList`, `LinkedList`, `Vector`, `Stack` |
| `Set` | `HashSet`, `LinkedHashSet`, `TreeSet` |
| `Queue` | `PriorityQueue`, `ArrayDeque` |
| `Map` | `HashMap`, `TreeMap`, `LinkedHashMap` |

# List

Η δομή αυτή είναι κατάλληλη για σειριακή αποθήκευση. Τα στοιχεία αποθηκεύονται σε θέσεις: 0, 1, ... κλπ. Η `ArrayList` είναι μια δομή αυτής της κατηγορίας, η `LinkedList` άλλη μια.

# ArrayList

Χρησιμοποιεί έναν δυναμικό πίνακα για να αποθηκεύσει τα στοιχεία του, χωρίς περιορισμό μεγέθους (αντίθετα από τους πίνακες - arrays).

- Μπορεί να περιέχει διπλότυπα,
- Η σειρά εισαγωγής διατηρείται,
- Παρέχει πρόσβαση με βάση τη θέση (index),
- Είναι λίγο πιο αργό από τη `LinkedList` γιατί χρειάζεται να αναδιαταχθούν τα στοιχεία του όταν έχουμε αφαίρεση.

# ArrayList (2)

```
ArrayList<String> list=new ArrayList<String>();

list.add("Μήλο");
list.add("Μπανάνα");
System.out.println(list);

Iterator iterator = list.iterator();

while(iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

# ArrayList (3)

```
ArrayList<String> list=new ArrayList<String>();

list.add("Μήλο");
list.add("Μπανάνα");

list.set(0, "Πορτοκάλι"); //αλλαγή του στοιχείου

Collections.sort(list); //ταξινόμιση

for(String fruit:list)
    System.out.println(fruit);
```

# Generics

Μια λίστα (που κληρονομεί από τη `List`) μπορεί να περιέχει αντικείμενα οποιουδήποτε τύπου (δηλ. `Object`):

```
List list = new ArrayList();

list.add(1);
list.add("ένα");
```

Συνεπώς χρειάζεται να γίνει casting όταν ανακτούμε το αντικείμενο:

```
Integer integer = (Integer) list.get(0);
String string   = (String) list.get(1);

//σφάλμα χρόνου μεταγλώττισης
String string2  = list.get(1);
```

# Generics (2)

Με τα generics είναι δυνατό να περιορίσουμε τον τύπο των αντικειμένων:

```
List<String> friendsNames = new ArrayList<String>();
// ή, μπορεί να μαντέψει τον τύπο από τον τύπο της μεταβλητής
List<String> friendsNames = new ArrayList<>();

friendsNames.add("Μάρκος");

String aFriend = friendsNames.get(0)
```

## Πλεονεκτήματα

- Ασφάλεια τύπου. Αντικείμενα μόνο ένας τύπου αποθηκεύονται.
- Δεν χρειάζεται casting όταν ανακτούμε αντικείμενο.
- Έλεγχος στο χρόνο μεταγλώττισης.

# LinkedList

Η `LinkedList` αποθηκεύει στοιχεία σε μια λίστα. Μπορούμε να εισάγουμε στοιχεία και από τις δύο άκρες τις λίστας.

Προτιμάται από την `ArrayList` σε σενάρια όπου χρειάζεται να μεταβάλλουμε τα δεδομένα που αποθηκεύουμε.

- Μπορεί να περιέχει διπλότυπα,
- Η σειρά εισαγωγής διατηρείται,
- Είναι γρήγορη γιατί δεν χρειάζεται να αναδιαταχθούν τα στοιχεία του,
- Μπορεί να χρησιμοποιηθεί σαν λίστα, στοίβα, ουρά.

# Set

Δομές για αποθήκευση όταν η σειρά δεν έχει σημασία. Δεν επιτρέπονται διπλές τιμές σε ένα `Set`.

Δομές αυτής της κατηγορίας είναι η `HashSet` και η `TreeSet`.

Ένα `Set` μπορεί να μας πει αν κάποια τιμή περιέχεται στα στοιχεία του ή όχι, αλλά δεν ξέρει τη σειρά με την οποία προστέθηκαν στοιχεία.

Για παράδειγμα αν θέλουμε να έχουμε μια λίστα με απαγορευμένες λέξεις, η σειρά δε θα έχει ιδιαίτερη σημασία.

# Set (παράδειγμα)

```
HashSet<String> badWords = new HashSet<>();

badWords.add("sex");
badWords.add("drugs");
badWords.add("rock 'n roll");
badWords.add("c++");

...

if (badWords.contains(word.toLowerCase()))
    System.out.println("Παρακαλώ διάλεξεάλληλέξη  ");
```

# Map

Ένα `Map` αποθηκεύει αντιστοιχίσεις μεταξύ κλειδιών και τιμών.

Με τη μέθοδο `put` προστίθεται μια νέα αντιστοίχιση (δηλ. ένα νέο ζεύγος κλειδί-τιμή) ή αλλάζει η τιμή ενός υπάρχοντος κλειδιού:

```
HashMap<String, String> friendsLocation = new HashMap<>();

// προσθέτει το ζεύγος κλειδί-τιμή στο Map
friendsLocation.put("Παύλος", "Αθήνα");

// ενημερώνει την τιμή για το κλειδί "Παύλος"
friendsLocation.put("Παύλος", "Ρώμη");
```

# Map (2)

Για να ανακτήσουμε την τιμή:

```
    Sting location = friendsLocation.get("Παύλος");
```

Αν δεν υπάρχει το κλειδί, τότε η `get()` θα επιστρέψει `null`. Στο παράδειγμά μας θα είχαμε μια εξαίρεση τύπου `NullPointerException` όταν θα προσπαθήσουμε να αναθέσουμε την τιμή `null` στη μεταβλητή `friendsLocation`.

Για να πάρουμε την τιμή `"unknown"` αν δεν υπάρχει το κλειδί στο `Map`, μπορούμε να γράψουμε το εξής:

```
    String location = friendsLocation.getOrDefault("Παύλος", "
        unknown");
```

# Queue

Δομή όπου η σειρά εισαγωγής έχει σημασία. Σε μια `Queue` προσθέτουμε αντικείμενα μόνο στο τέλος και μπορούμε να τα αφαιρέσουμε μόνο από την αρχή (σαν μια ουρά στην τράπεζα).

Μια δομή `Deque` (double-ended queue) επιτρέπει την εισαγωγή και αφαίρεση και από τα δύο άκρα.

# Stack

Μια αντίστοιχη δομή του Collection Framework είναι το `Deque` interface (με υλοποίηση στην `ArrayDeque`).

Η δομή `Stack` επιτρέπει την εισαγωγή και την αφαίρεση μόνο στο ένα άκρο (στοίβα).

```
    Stack<String> stack = new Stack<>();

    stack.push("Πέτρος");
    stack.push("Παύλος");
    stack.push("Μαρία");

    while (!stack.isEmpty())
        System.out.println(stack.pop());
```