

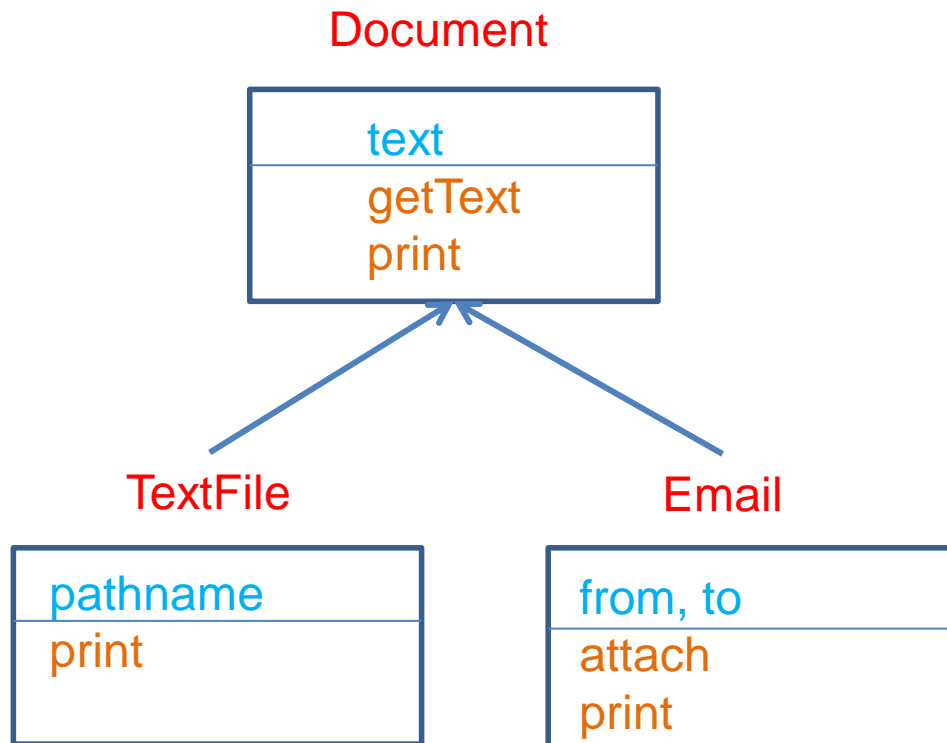
ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Παράδειγμα Κληρονομικότητας

Γενικευμένες κλάσεις
Συλλογές

Από το προηγούμενο lab

- Είχαμε την κλάση **Document**, και δύο παραγόμενες κλάσεις: **TextFile**, **Email**



Η ερώτηση ζητούσε να φτιάξετε ένα πίνακα με τρία **Document** αντικείμενα, ένα **Document**, ένα **TextFile**, και ένα **Email** και να κάνετε **attach** το **TextFile** σε ένα **Email**.

Τι γίνεται αν χρησιμοποιήσουμε αυτό τον κώδικα?

```
public class Test2
{
    public static void main(String args[]){
        Document[] documents = new Document[3];
        documents[0] = new Document("This is a document");
        documents[1] = new Email("Alice", "Bob", "Alice to Bob email");
        documents[2] = new TextFile("/home/myfile.txt", "This is a text file");
        documents[1].attach(documents[2]);
        printDocuments(documents);
    }

    private static void printDocuments(Document[] docs){
        for (int i = 0; i < docs.length; i ++){
            docs[i].print();
            System.out.println();
        }
    }
}
```

Χτυπάει **λάθος** γιατί η κλάση Document δεν έχει μέθοδο attach

Το Late Binding δεν έχει σχέση σε αυτή την περίπτωση γιατί η μέθοδος attach δεν υπάρχει στην Document, άρα ο κώδικας δεν περνάει το compilation.

Λύση: Downcasting

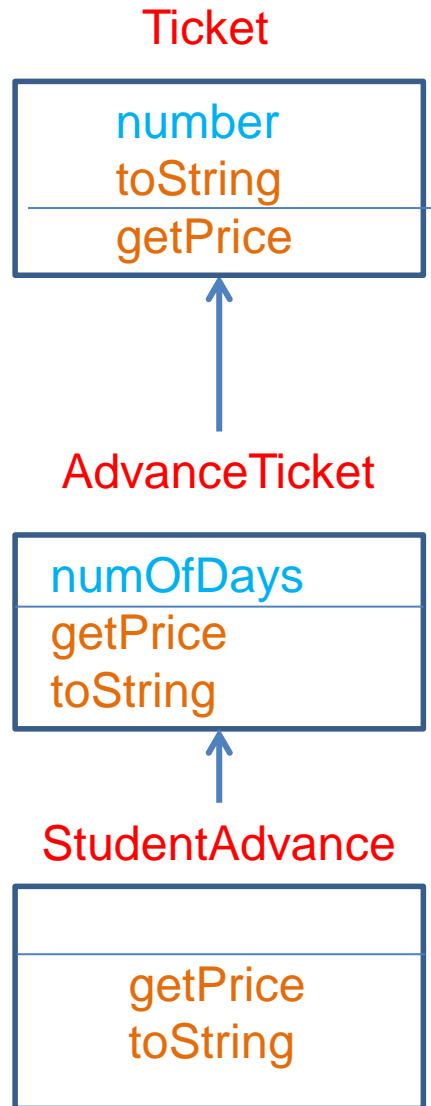
```
public class Test2
{
    public static void main(String args[]){
        Document[] documents = new Document[3];
        documents[0] = new Document("This is a document");
        documents[1] = new Email("Alice", "Bob", "Alice to Bob email");
        documents[2] = new TextFile("/home/myfile.txt", "This is a text file");
        Email myEmail = (Email) documents[1];
        myEmail.attach(documents[2]);
        documents[1] = myEmail;
        printDocuments(documents);
    }

    private static void printDocuments(Document[] docs){
        for (int i = 0; i < docs.length; i ++){
            docs[i].print();
            System.out.println();
        }
    }
}
```

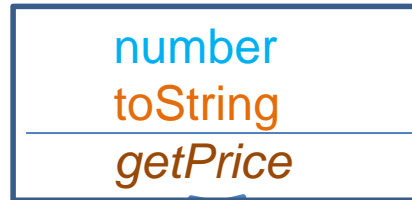
Άλλο ένα παράδειγμα

- Έχουμε ένα σύστημα διαχείρισης **εισιτηρίων** μιας συναυλίας. Το κάθε εισιτήριο έχει ένα **νούμερο** και **τιμή**. Η τιμή του εισιτηρίου εξαρτάται αν θα αγοραστεί στην **είσοδο** (50 ευρώ), ή θα αγοραστεί μέχρι και **10 μέρες πριν την συναυλία** (40 ευρώ), ή **πάνω από 10 μέρες πριν την συναυλία** (30 ευρώ). Τα εισιτήρια εκ των προτέρων έχουν **φοιτητική έκπτωση 50%**.
- Θέλουμε να **τυπώσουμε τα εισιτήρια** και να **υπολογίσουμε τα συνολικά έσοδα** της συναυλίας.

Ένας σχεδιασμός



Ticket

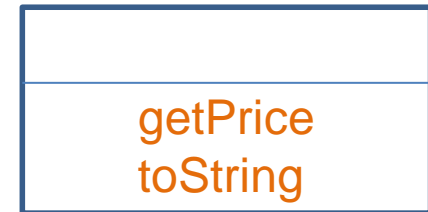


Ένας άλλος σχεδιασμός

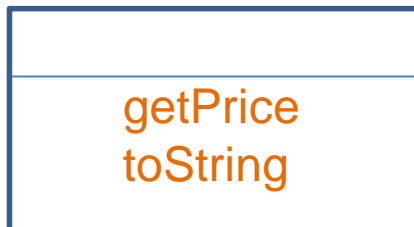
AdvanceTicket



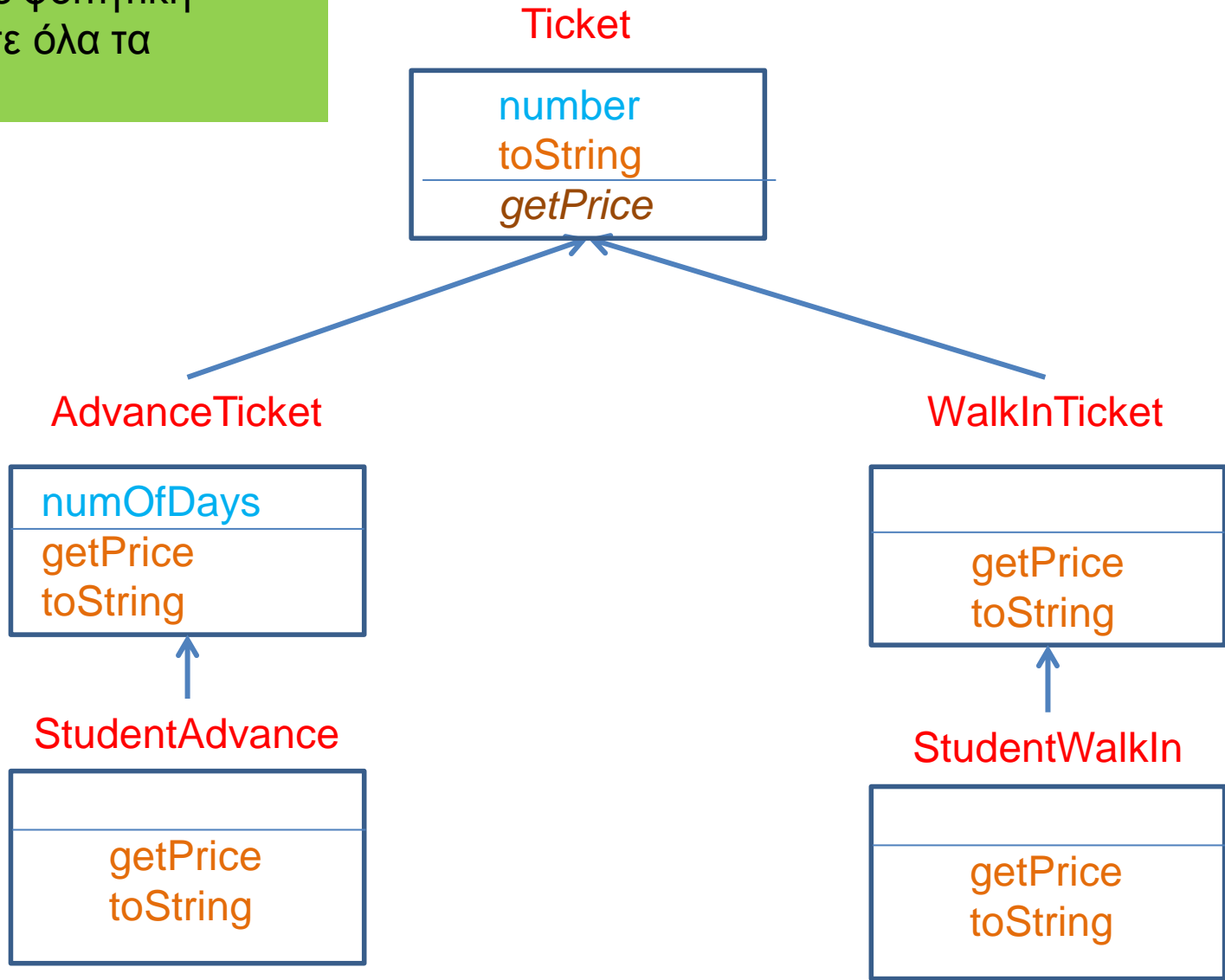
WalkInTicket



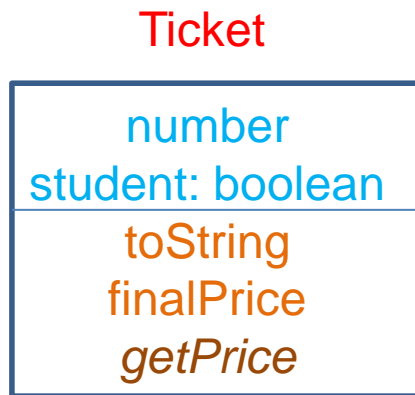
StudentAdvance



Αν θέλουμε φοιτητική έκπτωση σε όλα τα εισιτήρια?

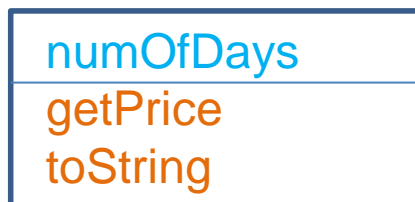


Αν θέλουμε φοιτητική έκπτωση σε όλα τα εισιτήρια?

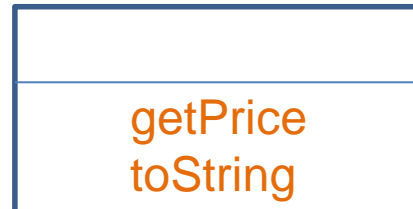


```
public abstract double getPrice();  
  
public double finalPrice()  
{  
    if (student){  
        return getPrice()*0.5;  
    }  
    return getPrice();  
}
```

AdvanceTicket



WalkInTicket



ΓΕΝΙΚΕΥΜΕΝΕΣ ΚΛΑΣΕΙΣ

Stack

- Θυμηθείτε πως ορίσαμε μια **στοίβα ακεραίων**

```
public class IntStack
{
    private IntStackElement head;
    private int size = 0;

    public int pop() {
        if (size == 0) { // head == null
            System.out.println("Pop from empty stack");
            System.exit(-1);
        }
        int value = head.getValue();
        head = head.getNext();
        size--;
        return value;
    }

    public void push(int value) {
        IntStackElement element = new IntStackElement(value);
        element.setNext(head);
        head = element;
        size++;
    }
}
```

```
public class IntStackElement
{
    private int value;
    private IntStackElement next = null;

    public IntStackElement(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }

    public IntStackElement getNext() {
        return next;
    }

    public void setNext(IntStackElement element) {
        next = element;
    }
}
```

Stack

- Αν θέλουμε η **στοίβα** μας να αποθηκεύει αντικείμενα της κλάσης **Person** θα πρέπει να ορίσουμε μια διαφορετική **Stack** και διαφορετική **StackElement**.

```
class PersonStackElement
{
    private Person value;
    private PersonStackElement next;

    public PersonStackElement(Person val) {
        value = val;
    }

    public void setNext(PersonStackElement element) {
        next = element;
    }

    public PersonStackElement getNext() {
        return next;
    }

    public Person getValue() {
        return value;
    }
}
```

```
public class PersonStack
{
    private PersonStackElement head;
    private int size = 0;

    public Person pop() {
        if (size == 0) { // head == null
            System.out.println("Pop from empty stack");
            System.exit(-1);
        }
        Person value = head.getValue();
        head = head.getNext();
        size--;
        return value;
    }

    public void push(Person value) {
        PersonStackElement element = new PersonStackElement(value);
        element.setNext(head);
        head = element;
        size++;
    }
}
```


Stack

- Θα ήταν πιο βολικό αν μπορούσαμε να ορίσουμε **μία μόνο** κλάση **Stack** που να μπορεί να αποθηκεύει αντικείμενα οποιουδήποτε τύπου.
 - **Πώς** μπορούμε να το κάνουμε αυτό?
- Μια λύση: Η **ObjectStack** που κρατάει αντικείμενα **Object**, την πιο γενική κλάση
- Τι πρόβλημα μπορεί να έχει αυτό?

```
class ObjectStackElement
{
    private Object value;
    private ObjectStackElement next;

    public ObjectStackElement(Object val) {
        value = val;
    }

    public void setNext(ObjectStackElement element) {
        next = element;
    }

    public ObjectStackElement getNext() {
        return next;
    }

    public Object getValue() {
        return value;
    }
}
```

```
public class ObjectStack
{
    private ObjectStackElement head;
    private int size = 0;

    public Object pop() {
        if (size == 0) { // head == null
            System.out.println("Pop from empty stack");
            System.exit(-1);
        }
        Object value = head.getValue();
        head = head.getNext();
        size--;
        return value;
    }

    public void push(Object value) {
        ObjectStackElement element = new ObjectStackElement(value);
        element.setNext(head);
        head = element;
        size++;
    }
}
```

```
public class ObjectStackTest
{
    public static void main(String[] args) {
        ObjectStack stack = new ObjectStack();

        Person p = new Person("Alice", 1);
        Integer i = new Integer(10);
        String s = "a random string";

        stack.push(p);
        stack.push(i);
        stack.push(s);
    }
}
```

Δεν μπορούμε να **ελέγξουμε** τι αντικείμενα μπαίνουν στην στοίβα. Κατά την εξαγωγή θα πρέπει να γίνει **μετατροπή (downcasting)** και θέλει προσοχή να μετατρέπουμε το σωστό αντικείμενο στον σωστό τύπο.

Θέλουμε να δημιουργούμε στοίβες **συγκεκριμένου τύπου**.

Γενικευμένες (Generic) κλάσεις

- Οι γενικευμένες κλάσεις περιέχουν ένα τύπο δεδομένων **T** που ορίζεται **παραμετρικά**
- Όταν χρησιμοποιούμε την κλάση αντικαθιστούμε την παράμετρο **T** με τον **τύπο δεδομένων** (την κλάση) που θέλουμε
- ΣΥΝΤΑΚΤΙΚΟ:
 - `public class Example<T> {...}`
- Ορίζει την γενικευμένη κλάση `Example` με παράμετρο τον τύπο **T**
 - Μέσα στην κλάση ο τύπος **T** χρησιμοποιείται σαν **τύπος δεδομένων**
- Όταν χρησιμοποιούμε την κλάση `Example` αντικαθιστούμε το **T** με κάποια συγκεκριμένη **κλάση**
 - `Example<String> ex = new Example<String>();`

Ένα πολύ απλό παράδειγμα

```
public class Example<T>{
    T data;

    public Example(T data){
        this.data = data;
    }

    public void setData(T data){
        this.data = data;
    }

    public T getData(){
        return data;
    }

    public static void main(String[] args){
        Example<String> ex = new Example<String>("hello world");
        System.out.println(ex.getData());
    }
}
```

Όταν ορίζουμε το αντικείμενο `ex` η κλάση `String` αντικαθιστά τις εμφανίσεις του `T` στον κώδικα

Ο ορισμός του constructor γίνεται χωρίς το `<T>` παρότι στην δημιουργία του αντικειμένου χρησιμοποιούμε το `<String>`

Γενικευμένη Στοίβα

- Μπορούμε τώρα να φτιάξουμε μια στοίβα για οποιοδήποτε τύπο δεδομένων

```
class StackElement<T>
{
    private T value;
    private StackElement<T> next;

    public StackElement(T val) {
        value = val;
    }

    public void setNext(StackElement<T> element) {
        next = element;
    }

    public StackElement<T> getNext() {
        return next;
    }

    public T getValue() {
        return value;
    }
}
```



```
public class Stack<T>
{
    private StackElement<T> head;
    private int size = 0;

    public T pop(){
        if (size == 0){ // head == null
            System.out.println("Pop from empty stack");
            System.exit(-1);
        }
        T value = head.getValue();
        head = head.getNext();
        size --;
        return value;
    }

    public void push(T value){
        StackElement<T> element = new StackElement<T>(value);
        element.setNext(head);
        head = element;
        size ++;
    }
}
```

```
public class StackTest
{
    public static void main(String[] args) {
        Stack<Person> personStack = new Stack<Person>();

        personStack.push(new Person("Alice", 1));
        personStack.push(new Person("Bob", 2));
        System.out.println(personStack.pop());
        System.out.println(personStack.pop());

        Stack<Integer> intStack = new Stack<Integer>();
        intStack.push(new Integer(10));
        intStack.push(new Integer(20));
        System.out.println(intStack.pop());
        System.out.println(intStack.pop());

        Stack<String> stringStack = new Stack<String>();
        stringStack.push("string 1");
        stringStack.push("string 2");
        System.out.println(stringStack.pop());
        System.out.println(stringStack.pop());
    }
}
```

Δημιουργούμε στοίβες **συγκεκριμένου τύπου**.

Πολλαπλές παράμετροι

- Μπορούμε να έχουμε πάνω από ένα παραμετρικούς τύπους

```
public class KeyValuePair<K, V>{  
    private K key;  
    private V value;  
    ...  
}
```

Παγίδες

1. Ο τύπος **T** **δεν** μπορεί να αντικατασταθεί από ένα **πρωταρχικό τύπο δεδομένων** (π.χ. int, double, boolean – πρέπει να χρησιμοποιήσουμε τα **wrapper classes** γι αυτά, Integer, Boolean, Double)
2. **Δεν** μπορούμε να ορίσουμε ένα **πίνακα** από αντικείμενα γενικευμένης κλάσης.

Π.χ., `StackElement<String>[] A;`

Δεν είναι αποδεκτό!

3. **Δεν** μπορούμε να χρησιμοποιούμε τον τύπο **T** όπως οποιαδήποτε άλλη κλάση.

Π.χ., `T obj = new T();`
`T[] a = new T[10];`

Δεν είναι αποδεκτό!

Γενικευμένες κλάσεις με περιορισμούς

- Ας υποθέσουμε ότι θέλουμε να ορίσουμε μία γενικευμένη κλάση `Pair` η οποία κρατάει ένα ζεύγος από δυο αντικείμενα οποιουδήποτε τύπου.

```
public class Pair<T>{  
    private T first;  
    private T second;  
    ...  
}
```

Γενικευμένες κλάσεις με περιορισμούς

- Θέλουμε επίσης να μπορούμε να **διατάσουμε** τα ζεύγη
 - Για να γίνει αυτό θα πρέπει να υπάρχει τρόπος να **συγκρίνουμε** τα στοιχεία first και second.
 - **Περιορίζουμε** την T να **υλοποιεί** το **interface myComparable**

```
public class Pair<T extends myComparable>{  
    private T first;  
    private T second;  
  
    public void order(){  
        if (first.compareTo(second) > 0){  
            T temp = first; first = second; second = temp;  
        }  
    }  
}
```

extends όχι implements

Γενικευμένες κλάσεις με περιορισμούς

- Θέλουμε επίσης να μπορούμε να **διατάσουμε** τα ζεύγη
 - Για να γίνει αυτό θα πρέπει να υπάρχει τρόπος να **συγκρίνουμε** τα στοιχεία first και second.
 - **Περιορίζουμε** την T να **υλοποιεί** το **interface Comparable**

```
public class Pair<T extends Comparable<T>>{  
    private T first;  
    private T second;  
  
    public void order(){  
        if (first.compareTo(second) > 0){  
            T temp = first; first = second; second = temp;  
        }  
    }  
}
```

Η **Comparable<T>** της Java
Το **T** είναι ο τύπος με τον οποίο
μπορούμε να συγκρίνουμε

Γενικευμένες κλάσεις με περιορισμούς

- Μπορούμε να περιορίσουμε τον παραμετρικό τύπο να κληρονομεί οποιαδήποτε κλάση, ή οποιοδήποτε interface ή συνδυασμό από τα παραπάνω.

- `public class SomeClass`

- `<T extends Employee> { ... }`

Δέχεται μόνο απογόνους της Employee

- `public class SomeClass`

- `<T extends Employee & Comparable<T>>`

- `{ ... }`

Δέχεται μόνο απογόνους της Employee που υλοποιούν το interface Comparable

Γενικευμένες κλάσεις και κληρονομικότητα

- Μια γενικευμένη κλάση μπορεί να έχει απογόνους άλλες γενικευμένες κλάσεις.
 - Οι απόγονοι κληρονομούν και τον τύπο T.
 - `public class OrderedPair<T> extends Pair<T> { ... }`
- Δεν ορίζεται κληρονομικότητα ως προς τον παραμετρικό τύπο T
 - Δεν υπάρχει καμία σχέση μεταξύ των κλάσεων `Pair<Employee>` και `Pair<HourlyEmployee>`

Wildcard

- Αν θέλουμε να ορίσουμε ένα γενικό παραμετρικό τύπο χρησιμοποιούμε την παράμετρο μπαλαντέρ ?, η οποία αναπαριστά ένα οποιοδήποτε τύπο T.
 - Προσέξτε ότι αυτό είναι κατά τη χρήση της γενικευμένης κλάσης
- `public void someMethod(Pair<?>) { ... }`
 - Με αυτή τη δήλωση ορίζουμε μία μέθοδο που παίρνει σαν όρισμα ένα αντικείμενο Pair με τύπο T οτιδήποτε.
- Μπορούμε να περιοριστούμε σε ένα τύπο που είναι απόγονος της Employee.
- `public void someMethod(Pair<? extends Employee>) { ... }`