

ΕΣΩΤΕΡΙΚΕΣ ΚΛΑΣΕΙΣ

Ποιά από τα πιο κάτω περιγράφουν όλα τα έγκυρα μέλη μιας κλάσης Java;

- Μη στατικά πεδία δεδομένων, μέθοδοι στιγμιοτύπου, στατικά πεδία δεδομένων, στατικές μέθοδοι
- Δημόσια πεδία δεδομένων, δημόσιες μέθοδοι στιγμιοτύπου
- Πεδία δεδομένων και μέθοδοι, κλάσεις, διαπροσωπείες
- Πεδία δεδομένων και μέθοδοι

1

ΕΠΑ233

2

Μέλη κλάσεων

- Τι είδους είναι τα μέλη μιας κλάσης;
 - Πεδία δεδομένων:
 - Μη στατικά: instance data fields or instance variables
 - Στατικά πεδία δεδομένων: static fields or class variables
 - Μέθοδοι
 - Μη στατικές: instance methods
 - Στατικές: class methods

Εσωτερικές Κλάσεις

- Στην Java είναι δυνατόν να τοποθετήσουμε τον ορισμό μιας κλάσης μέσα στον ορισμό κάποιας άλλης κλάσης.
- Οι κλάσεις που δηλώνονται κατ' αυτό τον τρόπο λέγονται **εσωτερικές (inner classes)**.
 - Οι εσωτερικές κλάσεις μας επιτρέπουν να ομαδοποιούμε πεδία δεδομένων τα οποία είναι συναφή λογικά και να ελέγχουμε από κοινού την πρόσβαση σε αυτά.
 - Ο ορισμός εσωτερικών κλάσεων σε μια κλάση A, **δεν συνεπάγεται** και τη δημιουργία αντίστοιχων υποαντικειμένων όταν δημιουργούμε αντικείμενα της κλάσης A.
- Με άλλα λόγια, **οι εσωτερικές κλάσεις είναι ένας τρόπος δημιουργίας νέων τύπων, μέσα σε υπάρχουσες κλάσεις.**
- Όπως συμβαίνει για όλα τα 'μέλη' μιας κλάσης, έτσι και για τις εσωτερικές κλάσεις μπορούμε να καθορίζουμε την εμβέλεια τους σαν δημόσια, ιδιωτική, προφυλαγμένη ή φιλική.

Εσωτερικές Κλάσεις - παράδειγμα

```
public class Parcel1 {  
    class Contents {  
        private int i = 11;  
        public int value() {return i; }  
    }  
    class Destination {  
        private String label;  
        Destination(String whereTo) {  
            label = whereTo;  
        }  
        String readLabel() { return label; }  
    }  
}
```

Εσωτερικές Κλάσεις - παράδειγμα

```
public void ship(String dest) {  
    Contents c = new Contents();  
    Destination d = new Destination(dest);  
    System.out.println(d.readLabel());  
}  
  
public static void main(String[] args) {  
  
    Parcel1 p = new Parcel1();  
    p.ship("Tanzania");  
    }  
}
```

Δημιουργία Αντικειμένων Εσωτερικών Κλάσεων

- Η δημιουργία αντικειμένου εσωτερικής κλάσης σε σημείο εκτός της περικλείουσας κλάσης, γίνεται μέσω **αντικειμένου** της περικλείουσας κλάσης:

```
Parcel1.Contents c = p.new Contents();
```

Επιστροφή αναφοράς σε εσωτερική κλάση

```
public class Parcel2 {  
    class Contents {  
        private int i = 11;  
        public int value() { return i; }  
    }  
    class Destination {  
        private String label;  
        Destination(String whereTo) { label = whereTo; }  
        String readLabel() { return label; }  
    }  
    public Destination to(String s) {  
        return new Destination(s); }  
    public Contents cont() { return new Contents(); }  
    public void ship(String dest) {  
        Contents c = cont();  
        Destination d = to(dest);  
        System.out.println(d.readLabel()); }  
}
```

Επιστροφή αναφοράς σε εσωτερική κλάση

```
public static void main(String[] args) {  
    Parcel2 p = new Parcel2();  
    p.ship("Tanzania");  
    Parcel2 q = new Parcel2();  
    // Defining references to inner classes:  
    Parcel2.Contents c = q.cont();  
    Parcel2.Destination d = q.to("Borneo");  
}
```

Για την δήλωση ενός αντικειμένου μιας εσωτερικής κλάσης, οπουδήποτε αλλού εκτός από το εσωτερικό μιας μη-στατικής μεθόδου της εξωτερικής κλάσης της, πρέπει να χρησιμοποιείται ο τύπος του αντικειμένου ως εξής: OuterClassName.InnerClassName

Εσωτερικές κλάσεις και upcasting

- Σύνοψη:
 - Οι εσωτερικές κλάσεις μπορούν να χρησιμοποιηθούν ως μηχανισμός «απόκρυψης» ορισμένων μελών (πεδίων δεδομένων και μεθόδων) μιας κλάσης.
 - Ωστόσο, υπάρχει και άλλος απλούστερος τρόπος για «απόκρυψη» ορισμένων μελών μιας κλάσης, με τον χαρακτηρισμό τους ως «φιλικών» (ή ιδιωτικών).
- Επομένως, η βασική χρησιμότητα των εσωτερικών κλάσεων είναι άλλη. Η χρήση των εσωτερικών κλάσεων μας επιτρέπει τα εξής:
 - Αφού έχουμε ορίσει κάποια διαπροσωπεία..
 - .. να μπορούμε να κάνουμε αναβάθμιση αντικειμένων (upcasting) στη διαπροσωπεία αυτή, **αποκρύπτοντας ταυτόχρονα την υλοποίηση της κλάσης του αντικειμένου το οποίο αναβαθμίζουμε.**
- Θυμηθείτε ότι η υλοποίηση μιας διαπροσωπείας περιέχει κατ' ανάγκη **δημόσιες** μεθόδους. Επομένως, υπό κανονικές συνθήκες, δεν είναι δυνατή η απόκρυψη των δημόσιων μεθόδων.

Παράδειγμα

```
// Destination.java  
  
public interface Destination {  
    String readLabel();  
}  
  
// Contents.java  
  
public interface Contents {  
    int value();  
}
```

Παράδειγμα (συνέχεια)

```
// Parcel3.java Returning a reference to an inner class.  
public class Parcel3 {  
    private class PContents implements Contents {  
        private int i = 11;  
        public int value() { return i; }  
    }  
    protected class PDestination implements Destination {  
        private String label;  
        private PDestination(String whereTo) { label = whereTo; }  
        public String readLabel() { return label; }  
    }  
    public Destination dest(String s) {  
        return new PDestination(s);  
    }  
    public Contents cont() { return new PContents(); }  
}
```

Παράδειγμα (συνέχεια)

```
// TestParcel.java

public class TestParcel {

    public static void main(String[] args) {

        Parcel3 p = new Parcel3();

        Contents c = p.cont();           // upcasting

        Destination d = p.dest("Tanzania");

        Parcel3.PContents pc = p.new PContents();

    }

}
```

Σύνδεση με εξώτερη κλάση (παράδειγμα)

```
interface Selector {
    boolean end();
    Object current();
    void next();
}

public class Sequence {
    private Object[] objects;
    private int next = 0;
    public Sequence(int size) {
        objects = new Object[size];
    }
    public void add(Object x) {
        if(next < objects.length)
            objects[next++] = x;
    }
}
```

Σύνδεση με εξώτερη κλάση (παράδειγμα)

```
private class SSelector implements Selector {
    private int i = 0;
    public boolean end() { return i == objects.length; }
    public Object current() { return objects[i]; }
    public void next() { if(i < objects.length) i++; }
}

public Selector getSelector() { return new SSelector(); }

public static void main(String[] args) {

    Sequence sequence = new Sequence(10);
    for(int i = 0; i < 10; i++)
        sequence.add(Integer.toString(i));
    Selector selector = sequence.getSelector();
    while(!selector.end()) {
        System.out.println(selector.current());
        selector.next();
    }
} // end of Sequence class
```

Αναφορές στο πεδίο
objects της
περικλείουσας
κλάσης, γίνονται
χωρίς ιδιαίτερο
προσδιορισμό.

Λειτουργικότητα Sequence

- Ένας πίνακας σταθερού μεγέθους, «περιτυλιγμένος» μέσα στην κλάση Sequence
- Διαθέτει μέθοδο add() για πρόσθεση αντικειμένων στον πίνακα, εφόσον υπάρχει χώρος.
- Επιστρέφει ένα αντικείμενο τύπου Selector, μέσω του οποίου μπορούμε να διατρέξουμε τον πίνακα.
- Πρόκειται για ένα παράδειγμα του **σχεδιαστικού προτύπου επαναλήπτη** (Iterator design pattern), το οποίο θα δούμε και αργότερα.

Σύνδεση με εξώτερη κλάση

- Αντικείμενα εσωτερικών κλάσεων διαθέτουν αυτόματη πρόσβαση στα μέλη της περικλείουσας κλάσης. Έτσι, από ένα στιγμιότυπο μιας εσωτερικής κλάσης έχουμε δυνατότητα πρόσβασης σε μέλη του περικλείοντος αντικειμένου, χωρίς ιδιαίτερους προσδιορισμούς.
- Αυτό συμβαίνει διότι όταν δημιουργούμε ένα στιγμιότυπο εσωτερικής κλάσης, **το αντικείμενο αυτό διαθέτει υπόρρητο σύνδεσμο προς το αντικείμενο εκείνο το οποίο δίνει υπόσταση στην εσωτερική μας κλάση, και το οποίο ανήκει στην κλάση που περικλείει την εσωτερική μας κλάση.**
- Αυτό σημαίνει ότι:
 - Ένα αντικείμενο εσωτερικής κλάσης μπορεί να δημιουργηθεί μόνο σε συνδυασμό με αντικείμενο της περικλείουσας κλάσης της.
 - Σε εσωτερικές κλάσεις δεν μπορούμε να δηλώνουμε **στατικά** πεδία (δεδομένων, μεθόδους, εσωτερικές κλάσεις). **Γιατί;**
- Η υπόρρητη παραπομπή στο αντικείμενο της περικλείουσας κλάσης μπορεί να ανακτηθεί με χρήση του: **EnclosingClassName.this**

Σύνδεση με εξώτερη κλάση

- Η πρόσβαση προς τις εξώτερες κλάσεις, χωρίς ιδιαίτερους προσδιορισμούς, είναι δυνατή ακόμη και από πολλαπλώς φωλιασμένες εσωτερικές κλάσεις:

```
class MNA {  
    private void f() {}  
    class A {  
        private void g() {}  
        public class B {  
            void h() {  
                g();  
                f();  
            }  
        }  
    }  
}
```

```
public class MultiNestingAccess {  
    public static void main(String[] args) {  
        MNA mna = new MNA();  
        MNA.A mnaa = mna.new A();  
        MNA.A.B mnaab = mnaa.new B();  
        mnaab.h();  
    }  
}
```

«Τοπικές» εσωτερικές κλάσεις

- Εκτός από τη δήλωσή τους σαν πεδία κλάσεων, εσωτερικές κλάσεις μπορούν επίσης να ορισθούν και να δηλωθούν:
 - Μέσα στο σώμα μεθόδων.
 - Μέσα σε πεδία εμβέλειας (scopes) στο εσωτερικό μεθόδων.
 - Σαν **ανώνυμες κλάσεις** που υλοποιούν κάποια διαπροσωπεία (interface).
 - Σαν ανώνυμες κλάσεις που επεκτείνουν μια κλάση με μη προκαθορισμένο κατασκευαστή.
- Στόχος:
 - Υλοποίηση μιας διαπροσωπείας ώστε να μπορούμε να δημιουργήσουμε και να επιστρέψουμε χειριστήριο σε αντικείμενο της διαπροσωπείας
 - Δημιουργούμε μια κλάση που μας βοηθάει στην επίλυση ενός προβλήματος, αλλά την κρατάμε ιδιωτική

Τοπικές εσωτερικές κλάσεις

```
// Destination.java  
  
public interface Destination {  
    String readLabel();  
}  
  
// Contents.java  
  
public interface Contents {  
    int value();  
}
```

Τοπικές εσωτερικές κλάσεις

```
public class Parcel5 {
    public Destination dest(String s) {
        class PDestination implements Destination {
            private String label;
            private PDestination(String whereTo) {
                label = whereTo;
            }
            public String readLabel() { return label; }
        }
        return new PDestination(s);
    }
    public static void main(String[] args) {
        Parcel5 p = new Parcel5();
        Destination d = p.dest("Tanzania");
    }
}
```

Ποια είναι η εμβέλεια αυτού του ορισμού;

Εσωτερικές κλάσεις τοπικής εμβέλειας

```
public class Parcel6 {
    private void internalTracking(boolean b) {
        if(b) {
            class TrackingSlip {
                private String id;
                TrackingSlip(String s) {
                    id = s;
                }
                String getSlip() { return id; }
            }
            TrackingSlip ts = new TrackingSlip("slip");
            String s = ts.getSlip();
        }
        // Can't use it here! Out of scope:
        //!! TrackingSlip ts = new TrackingSlip("x");
    }
    public void track() { internalTracking(true); }
    public static void main(String[] args) {
        Parcel6 p = new Parcel6();
        p.track(); }
}
```

Ανώνυμες εσωτερικές κλάσεις

```
public class Parcel7 {
    public Contents cont() {
        return new Contents() {
            private int i = 11;
            public int value() { return i; }
        }; // Semicolon required in this case
    }
    public static void main(String[] args) {
        Parcel7 p = new Parcel7();
        Contents c = p.cont();
    }
}
```

Η μέθοδος cont() συνδυάζει τη δημιουργία της τιμής επιστροφής με τον ορισμό της κλάσης, η οποία αναπαριστά την τιμή επιστροφής.

Εσωτερική κλάση χωρίς όνομα: δημιουργία αντικειμένου της ανώνυμης κλάσης, η οποία κληρονομεί από την Contents

Ανώνυμες εσωτερικές κλάσεις

- Η μέθοδος cont() συνδυάζει τη δημιουργία αντικειμένου σαν τιμή επιστροφής με τον ορισμό της κλάσης στην οποία ανήκει το επιστρεφόμενο αντικείμενο.
- Η κλάση αυτή είναι **ανώνυμη**.
- Στον ορισμό της κλάσης δεν συμπεριλαμβάνουμε κατασκευαστή και συνεπώς η δημιουργία του αντίστοιχου αντικειμένου γίνεται με **υπόρρητη επίκληση** του προκαθορισμένου κατασκευαστή.
- Ωστόσο, υπάρχει δυνατότητα δήλωσης κατασκευαστή με παραμέτρους.

Ανώνυμες εσωτερικές κλάσεις και κατασκευαστές με παράμετρο

```
public class Wrapping {  
    private int i;  
    public Wrapping(int x) {  
        i = x;  
    }  
    public int value() {  
        return i;  
    }  
}
```

Ανώνυμες εσωτερικές κλάσεις και κατασκευαστές με παράμετρο

```
public class Parcel8 {  
    public Wrapping wrap(int x) {  
        // Base constructor call:  
        return new Wrapping(x) { // Pass constructor argument.  
            public int value() { return super.value() * 47; }  
            public int newValue() { return 80; }  
        }; // Semicolon required  
    }  
    public static void main(String[] args) {  
        Parcel8 p = new Parcel8();  
        Wrapping w = p.wrap(10);  
        System.out.println(w.value());  
        System.out.println(w.newValue());  
    }  
}
```

Εσωτερική κλάση χωρίς όνομα, η οποία είναι τύπου Wrapping, αλλά επεκτείνει τη Wrapping υπερσκελίζοντας τη value και προσθέτοντας τον ορισμό της newValue.

Αρχικοποιήσεις πεδίων εσωτερικών κλάσεων

- Κατά τον ορισμό των πεδίων μιας ανώνυμης κλάσης, επιτρέπεται η χρήση αρχικοποιητών, όπως και με τις κανονικές κλάσεις.
- Αν ωστόσο, σε κάποιο αρχικοποιητή θέλουμε να χρησιμοποιήσουμε αντικείμενο που ορίζεται εκτός της ανώνυμης εσωτερικής κλάσης, ο μεταγλωττιστής επιβάλλει όπως η παράμετρος με την οποία περνιέται αυτό το αντικείμενο είναι *τελική (final)*.

Αρχικοποιήσεις πεδίων εσωτερικών κλάσεων

```
public class Parcel9 {  
    // Argument must be final to use inside  
    // anonymous inner class:  
    public Destination dest(final String dest) {  
        return new Destination() {  
            private String label = dest; // initializer  
            public String readLabel() { return label; }  
        };  
    }  
    public static void main(String[] args) {  
        Parcel9 p = new Parcel9();  
        Destination d = p.dest("Tanzania");  
    }  
}
```

Αρχικοποιητής

Αρχικοποιήσεις πεδίων εσωτερικών κλάσεων

```
abstract class Base {
    public Base(int i) {
        System.out.println("Base constructor, i = " + i); }
    public abstract void f();
}

public class AnonymousConstructor {
    public static Base getBase(int i) {
        return new Base(i) {
            {
                System.out.println("Inside instance initializer");
            }
            public void f() { System.out.println("In anonymous f()"); }
        };
    }

    public static void main(String[] args) {
        Base base = getBase(47);
        base.f();
    }
}
```

Στατική μέθοδος getBase, η οποία επιστρέφει ανώνυμη εσωτερική κλάση.

Αρχικοποιητής στιγμιοτύπου (instance initializer), ο οποίος έχει ρόλο οιονεί κατασκευαστή για την ανώνυμη κλάση.

29

```
public class Parcel10 {
    public Destination
        destination(final String dest, final float price) {
        return new Destination() {
            private int cost;
            // instance initialisation for each object
            {
                cost = Math.round(price);
                if (cost > 100) System.out.println("Over budget");
            }
            private String label = dest;
            public String readLabel() { return label; }
        };
    }

    public static void main(String[] args) {
        Parcel10 p = new Parcel10();
        Destination d = p.destination("Tasma", 101.395F);
    }
}
```

30

Διαπροσωπείες και «Εργοστάσια» (Factories)

- Ένας τυπικός τρόπος για να κατασκευάζουμε αντικείμενα που υλοποιούν μια διαπροσωπεία είναι το σχεδιαστικό πρότυπο «Εργοστασιακή Μέθοδος» (Factory Method):
 - Αντί να καλούμε απευθείας έναν κατασκευαστή, καλούμε μια μέθοδο κατασκευής σε ένα αντικείμενο «εργοστάσιο» (factory object), η οποία μας επιστρέφει μια υλοποίηση της διαπροσωπείας.
 - Με τον τρόπο αυτό, θεωρητικά ο κώδικάς μας είναι πλήρως απομονωμένος από την υλοποίηση της διαπροσωπείας
 - Έτσι μπορούμε να ανταλλάξουμε μια υλοποίηση με μια άλλη χωρίς παρενέργειες.

```
interface Service {
    void method1();
    void method2();
}

interface ServiceFactory {
    Service getService();
}

class Implementation1 implements Service {
    Implementation1() {} // Package access
    public void method1() {print("Implementation1 method1");}
    public void method2() {print("Implementation1 method2");}
}

class Implementation1Factory implements ServiceFactory {
    public Service getService() {
        return new Implementation1();
    }
}
```



```

class Implementation2 implements Service {
    Implementation2() {} // Package access
    public void method1() {
        print("Implementation2 method1");
    }
    public void method2() {
        print("Implementation2 method2");
    }
}

class Implementation2Factory implements ServiceFactory {
    public Service getService() {
        return new Implementation2();
    }
}

```

**Αξιοποίηση εσωτερικών κλάσεων
στο Σχεδιαστικό Πρότυπο Εργοστασιακής
Μεθόδου**

```

public class Factories {

    public static void serviceConsumer(ServiceFactory fact) {
        Service s = fact.getService();
        s.method1();
        s.method2();
    }

    public static void main(String[] args) {
        serviceConsumer(new Implementation1Factory());
        // Implementations are completely interchangeable:
        serviceConsumer(new Implementation2Factory());
    }
}

```

33

34

Εσωτερικές κλάσεις και Factory Method

```

interface Service {
    void method1();
    void method2();
}

interface ServiceFactory { Service getService();}

class Implementation1 implements Service {
    private Implementation1() {}
    public void method1() {print("Implementation1 method1");}
    public void method2() {print("Implementation1 method2");}
    public static ServiceFactory factory =
        new ServiceFactory() {
            public Service getService() {
                return new Implementation1();
            }
        };
}

```

35

Μ. Δικαϊάκος, ΕΠΙΔ233

36

```

class Implementation2 implements Service {
    private Implementation2() {}
    public void method1() {print("Implementation2 method1");}
    public void method2() {print("Implementation2 method2");}
    public static ServiceFactory factory =
        new ServiceFactory() {
            public Service getService() {
                return new Implementation2();
            }
        };
}

public class Factories {
    public static void serviceConsumer(ServiceFactory fact) {
        Service s = fact.getService();
        s.method1();
        s.method2();
    }
    public static void main(String[] args) {
        serviceConsumer(Implementation1.factory);
        // Implementations are completely interchangeable:
        serviceConsumer(Implementation2.factory);
    }
}

```

37

Φωλιασμένες Κλάσεις (εσωτερικές στατικές)

- Μέχρι τώρα είδαμε ότι τις εσωτερικές κλάσεις τις διαχειριζόμαστε όπως και τα υπόλοιπα πεδία των κλάσεων της JAVA, όσον αφορά τον προσδιορισμό πρόσβασης προς αυτές (private κλπ).
- Η JAVA μας επιτρέπει να ορίζουμε εσωτερικές κλάσεις σαν **στατικές**. Οι στατικές εσωτερικές κλάσεις λέγονται **φωλιασμένες κλάσεις** (nested classes).
- Δημιουργία αντικειμένου φωλιασμένης κλάσης **δεν** προϋποθέτει την δημιουργία αντικειμένου εξωτερικής κλάσης.
- Φωλιασμένες κλάσεις μπορούν να ενυπάρχουν σαν πεδία *διαπρωπειών* (interfaces). Π.χ.:

```

public interface IInterface {
    static class Inner {
        int i, j, k;
        public Inner() {}
        void f() {}
    }
}

```

Μ. Δικαίος, ΕΠΙΔ233

Closure (κλείσιμο)

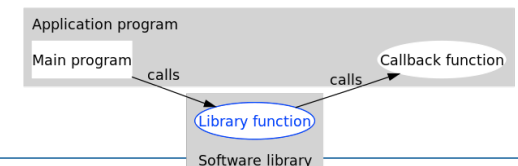
- **Closure** (κλείσιμο) αποκαλείται μια οντότητα η οποία διατηρεί πληροφορία σχετική με το πεδίο εμβέλειας στο οποίο δημιουργήθηκε.
- Στη Java, μια εσωτερική κλάση αποτελεί closure καθώς:
 - Παρ' ότι δεν περιλαμβάνει «αυτούσιες» πληροφορίες από το αντικείμενο το οποίο την περικλείει...
 - διατηρεί υπορρήτως μια παραπομπή στο αντικείμενο αυτό, μέσω της οποίας **διατηρεί τη δυνατότητα πρόσβασης σε όλα τα πεδία του αντικειμένου** (ακόμη και στα ιδιωτικά).
- Οι εσωτερικές κλάσεις της Java μας επιτρέπουν να υλοποιήσουμε **callbacks** (επανακλήσεις).

Μ. Δικαίος, ΕΠΙΔ233

39

Callback (επανάκληση)

- In computer programming, a callback is **a piece of executable code** that is passed as an argument to other code, which is expected to call back (execute) the argument at some convenient time.
 - The invocation may be immediate as in a synchronous callback, or it might happen at later time as in an asynchronous callback.
 - In all cases, the **intention is to specify a function or subroutine as an entity that is, depending on the language, more or less similar to a variable**.
- Programming languages support callbacks in different ways, often implementing them with **subroutines**, **lambda expressions**, **blocks**, or **function pointers**.



Μ. Δικαίος, ΕΠΙΔ233

40

Επανακλήσεις στη Java



- In object-oriented programming languages without function-valued arguments, such as in Java before its 8 version, callbacks can be simulated by **passing an instance of an abstract class or interface, of which the receiver will call one or more methods, while the calling end provides a concrete implementation.**
- Such objects are effectively a bundle of callbacks, plus the data they need to manipulate. They are useful in implementing various design patterns such as Visitor, Observer, and Strategy.
- Με το callback, ένα αντικείμενο μπορεί να λάβει από ένα δεύτερο αντικείμενο μια πληροφορία, η οποία του επιτρέπει να καλέσει μέλη του πρώτου αντικειμένου κάποια στιγμή αργότερα.

Παράδειγμα χρήσης callback

- Αναζητούμε ένα τρόπο ώστε η go() της Caller να μπορέσει να χρησιμοποιήσει την increment() που έχει υλοποιηθεί στην MyIncrement.
 - Παρατηρείστε ότι στον κατασκευαστή της Caller δεν μπορούμε να περάσουμε αντικείμενο τύπου MyIncrement!

```
interface Incrementable { void increment(); }
class Caller {
    private Incrementable callbackReference;
    Caller(Incrementable cbh) { callbackReference = cbh; }
    void go() { callbackReference.increment(); }
}
```

```
class MyIncrement {
    void increment() { System.out.println("Other operation" ); }
    static void f(MyIncrement mi) { mi.increment(); }
}
```

Παράδειγμα χρήσης callback (συνέχεια)

- Προσέγγιση: κατασκευάζουμε την Callee2, ώστε να κληρονομεί την υλοποίηση της MyIncrement.
 - Με την ευκαιρία, τροποποιούμε και την υλοποίηση της κληρονομημένης increment() μέσα στην Callee2
- Προσθέτουμε στην Callee2 μια εσωτερική κλάση Closure που υλοποιεί τη διαπρωσωπεία Incrementable
 - Η κλάση αυτή πρέπει να περιέχει μέθοδο increment()
 - Φροντίζουμε ώστε η νέα increment() της Closure, να καλεί την increment() της Callee2 την οποία και θέλαμε να επαναχρησιμοποιήσουμε.
- Η μέθοδος getCallbackReference() της Callee2 κατασκευάζει και επιστρέφει το αντικείμενο Closure το οποίο συγκρατεί στο εσωτερικό του το συγκεκριμένο της Callee2

```
class Callee2 extends MyIncrement {
    private int i = 0;
    public void increment() {
        super.increment();
        i ++;
    }
    private class Closure implements Incrementable {
        public void increment() { Callee2.this.increment(); }
    }
    Incrementable getCallbackReference() {
        return new Closure();
    }
}
```

Εσωτερικές κλάσεις και Πλαίσια ελέγχου

- **Πλαίσια εφαρμογών** (application frameworks): κλάση ή σύνολο κλάσεων το οποίο έχει σχεδιασθεί για την επίλυση μιας συγκεκριμένης κατηγορίας προβλημάτων.
- Για την χρήση ενός πλαισίου εφαρμογών, κατά κανόνα κληρονομούμε από τις κλάσεις του πλαισίου εφαρμογών και υπερσκελίζουμε την υλοποίηση ορισμένων μεθόδων.
- Ο κώδικας που γράφουμε στις υπερσκελίζουσες μεθόδους προσαρμόζει τη γενική λύση που δίνεται από το πλαίσιο εφαρμογών στο συγκεκριμένο πρόβλημα που μας ενδιαφέρει να λύσουμε.
- Μια κατηγορία πλαισίων εφαρμογών είναι τα **πλαίσια ελέγχου**, τα οποία αφορούν στον έλεγχο *συμβάντων* (events).
- Συστήματα λογισμικού τα οποία διαχειρίζονται συμβάντα λέγονται **συστήματα ελκόμενα από συμβάντα** (event-driven systems). Τυπικό παράδειγμα τέτοιων συστημάτων είναι τα GUI.

Προγραμματισμός ελκόμενος από συμβάντα



- Event-driven programming is a programming paradigm in which the **flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads**.
 - Event-driven programming is the dominant paradigm used in graphical user interfaces and other applications that are centered on performing certain actions in response to user input.
- In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.
- Event-driven programs can be written in any programming language, although the task is easier in languages that provide high-level abstractions, such as closures.

Παράδειγμα Πλαισίου Ελέγχου

- Παράδειγμα: εκτέλεση συμβάντων, όταν αυτά καθίστανται έτοιμα προς εκτέλεση μετά την πάροδο κάποιου χρονικού διαστήματος.

```
public abstract class Event {  
    private long eventTime;  
    protected final long delayTime;  
    public Event(long delayTime) {  
        this.delayTime = delayTime;  
        start();  
    }  
    public void start() {  
        eventTime = System.currentTimeMillis() + delayTime;  
    }  
    public boolean isReady() {  
        return System.currentTimeMillis() >= eventTime;  
    }  
    public abstract void action();  
}
```

Παράδειγμα Πλαισίου Ελέγχου

- Ο κατασκευαστής αντικειμένων τύπου Event αρχικοποιεί την τιμή της χρονικής στιγμής κατά την οποία μπορεί να εκτελεσθεί το συμβάν, μετά τη δημιουργία του.
- Μετά καλεί την start, η οποία υπολογίζει τη στιγμή κατά την οποία το συμβάν πρέπει να λάβει χώρα.

```
public Event(long delayTime) {  
    this.delayTime = delayTime;  
    start();  
}
```

```

import java.util.*;
public class Controller {
    private List eventList = new ArrayList();
    public void addEvent(Event c) {
        eventList.add(c);
    }
    public void run() {
        while (eventList.size() > 0) {
            for (int i = 0; i < eventList.size(); i++) {
                Event e = (Event)eventList.get(i);
                if (e.isReady()) {
                    System.out.println(e);
                    e.action();
                    eventList.remove(i);
                }
            }
        }
    }
}

```

49

Πλαίσια Ελέγχου και Εσωτερικές κλάσεις

- Η χρήση εσωτερικών κλάσεων μας επιτρέπει:
 - Να τοποθετήσουμε ολόκληρη την υλοποίηση ενός πλαισίου ελέγχου σε μια μόνο κλάση, η οποία να εγκυβωτίζει όλη τη λειτουργικότητα που αφορά στο συγκεκριμένο πρόβλημα.
 - Να διατηρήσουμε την απλότητα της υλοποίησης, καθώς τα αντικείμενα των εσωτερικών κλάσεων διατηρούν πρόσβαση προς όλα τα μέλη της περικλείουσας κλάσης.
- Παράδειγμα
 - Υλοποίηση λογισμικού ελέγχου θερμοκηπίου.
 - Ελέγχει διαφορετικές λειτουργίες του θερμοκηπίου, όπου κάθε ενέργεια είναι διαφορετική (π.χ. άνοιγμα φωτός, νερού, θερμοστάτη κλπ).

Μ. Δικαϊάκος, ΕΠΙΔ233

50

Έλεγχος θερμοκηπίου

```

public class GreenhouseControls extends Controller {
    private boolean light = false;
    public class LightOn extends Event {
        public LightOn(long delayTime) {super(delayTime);}
        public void action() { light = true; }
        public String toString() { return "Light is on"; }
    }

    public class LightOff extends Event {
        public LightOff(long delayTime) { super(delayTime); }
        public void action() { light = false; }
        public String toString() { return "Light is off"; }
    }
}

```

Μ. Δικαϊάκος, ΕΠΙΔ233

51

Έλεγχος θερμοκηπίου

```

private boolean water = false ;
public class WaterOn extends Event {
    public WaterOn(long delayTime) { super(delayTime); }
    public void action() {
        water = true ;
    }
    public String toString() {
        return "Greenhouse water is on";
    }
}

public class WaterOff extends Event {
    public WaterOff(long delayTime) { super(delayTime); }
    public void action() {
        water = false ;
    }
    public String toString() { return "Greenhouse water is off" ;}
}

```

Μ. Δικαϊάκος, ΕΠΙΔ233

52

Έλεγχος θερμοκηπίου

```
private String thermostat = "Day" ;
public class ThermostatNight extends Event {
    public ThermostatNight(long delayTime){super(delayTime);}
    public void action() {thermostat = "Night" ;    }
    public String toString(){return "Thermostat on night set";}
}

public class ThermostatDay extends Event {
    public ThermostatDay(long delayTime) {super(delayTime); }
    public void action() { thermostat = "Day" ;    }
    public String toString() {return "Thermostat on day set";}
}
```

Έλεγχος θερμοκηπίου

```
// An example of an action() that inserts a
// new one of itself into the event list
public class Bell extends Event {
    public Bell( long delayTime) {
        super(delayTime);
    }
    public void action() {
        addEvent(new Bell(delayTime));
    }
    public String toString() { return "Bing!" ; }
}
```

Έλεγχος θερμοκηπίου

```
public class Restart extends Event {
    private Event[] eventList;
    public Restart(long delayTime, Event[] eventList) {
        super(delayTime);
        this.eventList = eventList;
        for (int i = 0; i < eventList.length; i++)
            addEvent(eventList[i]);
    }
    public void action() {
        for (int i = 0; i < eventList.length; i++) {
            eventList[i].start(); // Rerun each event
            addEvent(eventList[i]);
        }
        start(); // Rerun this Event
        addEvent( this );
    }
    public String toString() { return "Restarting system" ;    }
}
```

Έλεγχος θερμοκηπίου

```
public static class Terminate extends Event {
    public Terminate(long delayTime) {
        super(delayTime);
    }
    public void action() {
        System.exit(0);
    }
    public String toString() { return "Terminating!" ; }
}
```

Έλεγχος θερμοκηπίου

```
public class GreenhouseController {
    public static void main(String[] args) {
        GreenhouseControls gc = new GreenhouseControls();
        gc.addEvent(gc.new Bell(900));
        Event[] eventList = {
            gc.new ThermostatNight(0),
            gc.new LightOn(200),
            gc.new LightOff(400),
            gc.new WaterOn(600),
            gc.new WaterOff(800),
            gc.new ThermostatDay(1400)};
        gc.addEvent(gc.new Restart(2000, eventList));
        if (args.length == 1)
            gc.addEvent(gc.new
                Terminate(Integer.parseInt(args[0])));
        gc.run();    }}
```

Έλεγχος θερμοκηπίου

```
public class Terminate extends Event {
    public Terminate( long delayTime) {
        super (delayTime);
    }
    public void action() {
        System.exit(0);
    }

    public String toString() {
        return "Terminating" ;
    }
}
```