# ΕΣΩΤΕΡΙΚΕΣ ΚΛΑΣΕΙΣ

# Εσωτερικές Κλάσεις

ΕΠΛ233

- 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 Α, **δεν συνεπάγεται** και τη δημιουργία αντίστοιχων υποαντικειμένων όταν δημιουργούμε αντικείμενα της κλάσης Α.
- 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

ΕΠΛ233

```
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

ΕΠΛ233

```
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

ΕΠΛ233

- 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

ΕΠΛ233

```
  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");
  }
}
```

# Επιστροφή αναφοράς σε εσωτερική κλάση

ΕΠΛ233

```
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 εξώτερης κλάσης της, πρέπει να χρησιμοποιείται ο τύπος του αντικειμένου ως εξής: OuterClassName.InnerClassName

# Εσωτερικές κλάσεις και upcasting

ΕΠΛ233

- 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)

ΕΠΛ233

```
// 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)

ΕΠΛ233

```
// 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)

ΕΠΛ233

```
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)

ΕΠΛ233

```
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();
    }}
```

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

# Σύνδεση με εξώτερη κλάση

ΕΠΛ233

- 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

ΕΠΛ233

- 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

ΕΠΛ233

- 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.
  - κοκ

# «Τοπικές» εσωτερικές κλάσεις

ΕΠΛ233

```
public class Parcel4 {
  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) {
    Parcel4 p = new Parcel4();
    Destination d = p.dest("Tanzania");
  }
}
```

Ποια είναι η εμβέλεια αυτού του ορισμού;

# Εσωτερικές κλάσεις τοπικής εμβέλειας

ΕΠΛ233

```
public class Parcel5 {
  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) {
    Parcel5 p = new Parcel5();
    p.track();
  }
```

# Ανώνυμες εσωτερικές κλάσεις

ΕΠΛ233

```
public class Parcel6 {
  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) {
    Parcel6 p = new Parcel6();
    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

ΕΠΛ233

- Η μέθοδος cont( ) συνδυάζει τη δημιουργία αντικειμένου σαν τιμής 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

ΕΠΛ233

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

# Ανώνυμες εσωτερικές κλάσεις

ΕΠΛ233

```
public class Parcel7 {
  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) {
    Parcel7 p = new Parcel7();
    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

ΕΠΛ233

- 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

ΕΠΛ233

```
public class Parcel8 {
    // 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) {
    Parcel8 p = new Parcel8();
    Destination d = p.dest("Tanzania");
  }
}
```

# Αρχικοποιήσεις πεδίων εσωτερικών κλάσεων

ΕΠΛ233

```
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) {
      int ff;
      String s;
      {
        System.out.println("Inside instance initializer");
        i = 0;
        s = new String("foo");
      }
      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.

# Φωλιασμένες Κλάσεις

ΕΠΛ233

- 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 εξώτερης κλάσης.
- Φωλιασμένες κλάσεις μπορούν να ενυπάρξουν σαν πεδία *διαπροσωπειών (interfaces).* Π.χ.:

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

# Closures και Callbacks

ΕΠΛ233

- Closure καλείται μια οντότητα η οποία διατηρεί πληροφορία σχετική με το πεδίο εμβέλειας στο οποίο δημιουργήθηκε.
- Στη Java, η 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.
- 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

ΕΠΛ233

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

// If your class must implement increment() in
// some other way, you must use an inner class:
class Callee2 extends MyIncrement {
   private int i = 0;
   private void incr() { i++; System.out.println(i);}
   private class Closure implements Incrementable {
       public void increment() { incr(); }
   }
   Incrementable getCallbackReference() {
       return new Closure();  }
   }
```

# Παράδειγμα (συνέχεια)

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

public class Callbacks {
  public static void main(String[] args) {
    Callee2 c2 = new Callee2();
    MyIncrement.f(c2);
    Caller caller2 = new Caller(c2.getCallbackReference());
    caller2.go();
    caller2.go();
  }}
```

# Εσωτερικές κλάσεις και Πλαίσια ελέγχου

# Εσωτερικές κλάσεις και Πλαίσια ελέγχου


- Πλαίσια εφαρμογών (application frameworks): κλάση ή σύνολο κλάσεων το οποίο έχει σχεδιασθεί για την επίλυση μιας συγκεκριμένης κατηγορίας προβλημάτων.

# 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.

# 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.

# 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.
- Ειδική περίπτωση των πλαισίων εφαρμογών είναι τα πλαίσια ελέγχου, τα οποία αφορούν στον έλεγχο *συμβάντων (events).*

# 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.
- Ειδική περίπτωση των 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.

# Παράδειγμα Πλαισίου Ελέγχου

ΕΠΛ233

- 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 σώζει το χρόνο (από την στιγμή δημιουργίας του συμβάντος) κατά τον οποίο θέλουμε να μπορεί να εκτελεσθεί το συμβάν.
- Μετά καλεί την start, η οποία υπολογίζει τη στιγμή κατά την οποία το συμβάν πρέπει να λάβει χώρα.

```
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);
  }}}}}
```

# 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).

# Έλεγχος θερμοκηπίου

ΕΠΛ233

```
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"; }
   }
```

# Έλεγχος θερμοκηπίου

ΕΠΛ233

```
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" ;}
}
```

# Έλεγχος θερμοκηπίου

ΕΠΛ233

```
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

ΕΠΛ233

```
// 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

ΕΠΛ233

```
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

ΕΠΛ233

```
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

ΕΠΛ233

```
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

ΕΠΛ233

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

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