# Διάλεξη 19: Φωλιασμένες (Nested) και Εσωτερικές (Inner) κλάσεις

**Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:**

- Φωλιασμένες Κλάσεις
- Εσωτερικές Κλάσεις (Τοπικές και Ανώνυμες)
- Ταξινομία Κλάσεων
- Εκτύπωση Χαρακτηριστικών Κλάσης

**Διδάσκων: Παναγιώτης Ανδρέου**

# Παράδειγμα Event Handling με ΔΥΟ Listeners

```
    JButton jbtOK = new JButton("OK");
    add(jbtOK);
    // Register listeners
    OKListenerClass listener1 = new OKActionListener();
    jbtOK.addActionListener(listener1);

    OKListenerClass listener2 = new OKMouseListener();
    jbtOK.addMouseListener(listener2);
  }
  public static void main(String[] args) {
    JFrame frame = new HandleEvent();
    …
  }
}
class OKActionListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("OK button clicked");
  }
}
class OKMouseListener implements MouseListener {
  public void mouseClicked(MouseEvent arg0) {
    System.out.println("mouseClicked");
  }
  public void mouseEntered(MouseEvent arg0) {
    System.out.println("mouseEntered");
  }
}
```

**Listener1**
ActionListener

**Listener2**
MouseListener

# Προβλήματα Κλάσεων Listeners

- Στο προηγούμενο παράδειγμα έχουμε δηλώσει 2 listeners σαν δύο ξεχωριστές κλάσεις
- Οι κλάσεις αυτές θα χρησιμοποιηθούν μόνο μέσα στο συγκεκριμένο πρόγραμμα/κλάση που έχουμε αναπτύξει.
- **Ερώτηση:** Μήπως έχουν και άλλες κλάσεις πρόσβαση σε αυτές τις κλάσεις;
- **Απάντηση:** Δυστυχώς ΝΑΙ! (τουλάχιστον στο ίδιο πακέτο)

- **Ερώτηση:** Μπορώ να δηλώσω μία κλάση listener (ActionListener) που να την χρησιμοποιήσω με όλα τα αντικείμενα που έχω;
- **Απάντηση:** Δυστυχώς ΟΧΙ!
  Μπορώ όμως να υλοποιήσω την διαπροσωπεία ActionListener στο top-level επίπεδο και να διαχειριστώ όλα τα ActionEvents από όλα τα αντικείμενα

# Διαχείριση Events από πολλά αντικ. εντός κλάσης

```
import …
public class HandleMultipleComponents
        extends JFrame implements ActionListener{
  // Create two buttons
  JButton jbtOK = new JButton("OK");
  JButton jbtCancel = new JButton("Cancel");

  public HandleMultipleComponents() {
    setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
    add(jbtOK); add(jbtCancel);
    // Register class listener
    jbtOK.addActionListener(this);
    jbtCancel.addActionListener(this);
  }
  //Implement actionPerformed
  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == jbtOK)
      System.out.println("OK");
    else if (e.getSource() == jbtCancel)
      System.out.println("Cancel");

  public static void main(String[] args) {
    JFrame frame = new HandleMultipleComponents();
    …
  }
 }
}
```

Υλοποίηση Διαπροσωπείας ActionListener Πρέπει να υλοποιηθεί η μέθοδος actionPerformed()

Πέρασμα του αντικειμένου της κλάσης (this) σαν παραμέτρο

# Παρατηρήσεις

- Αν και η λύση της προηγούμενης διαφάνειας είναι ελκυστική ας αναλύσουμε τι συμβαίνει στις εξής περιπτώσεις:

1. Η διαπροσωπεία μας περιέχει ***n*** components
   Χρειαζόμαστε ***n*** if statements
2. Η διαπροσωπεία μας διαχειρίζεται ***m*** events
   Χρειαζόμαστε ***m*** μεθόδους διαχείρισης (actionPerformed)
3. Η διαπροσωπεία μας περιέχει *n* components και το καθένα εγείρει *m* events
   Χρειαζόμαστε ***m*** μεθόδους διαχείρισης (actionPerformed) και ο καθένας θα διαχειρίζεται ***n*** if statements (***n*** x ***m***)

- … και αυτό χωρίς τον κώδικα διαχείρισης (δηλ. τι θα γράψουμε μέσα στη μέθοδο)

- **Υπάρχει καλύτερη λύση; ΝΑΙ, τα nested (inner) classes.**

# Φωλιασμένες Κλάσεις (Nested Classes)

- Οι κλάσεις listener που υλοποιήσαμε έχουν σχεδιαστεί για να δημιουργηθούν διαχειριστές συμβάντων για διάφορα αντικείμενα (π.χ., για ένα κουμπί)
- Αυτές οι κλάσεις δεν θα χρησιμοποιηθούν από καμία άλλη κλάση είτε εντός ή εκτός εφαρμογής
- Για αυτό το λόγο είναι πιο σωστό αυτές οι κλάσεις να ισχύουν μόνο σε επίπεδο της κλάσης που έχουν οριστεί
- Αυτές οι κλάσεις αποτελούν «πεδία» των κλάσεων που έχουν οριστεί

- **Κλάσεις που ορίζονται μέσα σε μία άλλη κλάση → Φωλιασμένες Κλάσεις (Nested Classes)**

# Παράδειγμα Φωλιασμένων Κλάσεων

```
import …
public class NestedClassExample extends JFrame {
  // Create two buttons
  JButton jbtOK = new JButton("OK");
  JButton jbtCancel = new JButton("Cancel");

  class OKListenerClass implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      System.out.println("OK");}
  }

  class CanceListenerClass implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      System.out.println("Cancel");}
  }

  public NestedClassExample () {
    setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
    add(jbtOK);
    add(jbtCancel);
    jbtOK.addActionListener(new OKListenerClass());
    jbtCancel.addActionListener(new CanceListenerClass());
  }

  public static void main(String[] args) {
    NestedClassExample frame = new NestedClassExample(); …
}
```

**Παραδείγματα Φωλιασμένων Κλάσεων**

# Φωλιασμένες Κλάσεις - Ενθυλάκωση

- Οι εσωτερικές κλάσεις είναι ένας τρόπος δημιουργίας νέων τύπων, μέσα σε υπάρχουσες κλάσεις.
- Μας επιτρέπουν να απλοποιούμε προγράμματα με την «ομαδοποίηση» υψηλά συσχετιζόμενων κλάσεων.
- Μία φωλιασμένη κλάση μπορεί να έχει πρόσβαση στα αντικείμενα της εξωτερικής κλάσης που την περιέχει
  ➔ Δεν χρειάζεται να περνούμε σαν παράμετρο το αντικείμενο της εξωτερικής κλάσης στον κατασκευαστή της φωλιασμένης κλάσης
- **ΠΡΟΣΟΧΗ:** Ο ορισμός φωλιασμένων κλάσεων σε μια κλάση, δεν σημαίνει και δημιουργία αντίστοιχων υποαντικειμένων της φωλιασμένης κλάσης.

# Φωλιασμ.Κ.: Πρόσβαση στα μέλη εξωτερικ. κλάσης

```
import …
public class NestedClassExampleHandleMultipleComponents
        extends JFrame {
  JButton jbtOK = new JButton("OK");
  JButton jbtCancel = new JButton("Cancel");

  class MyListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      if(e.getSource()==jbtOK)
        System.out.println("OK Clicked");
      if(e.getSource()==jbtCancel)
        System.out.println("Cancel Clicked");
    }
  }
  public NestedClassExampleHandleMultipleComponents() {
setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
    add(jbtOK);
    add(jbtCancel);
    jbtOK.addActionListener(new MyListener());
    jbtCancel.addActionListener(new MyListener());
  }
public static void main(String[] args) {
  NestedClassExampleHandleMultipleComponents frame = new
NestedClassExampleHandleMultipleComponents();
    frame.setTitle("InnerMemberHandleMultipleComponents
");
  }
}
```

**Πρόσβαση στα αντικείμενα της εξωτερικής κλάσης (χωρίς αντικείμενο)**

**Χρήση του ίδιου listener class**

# Αρχικοποίηση Φωλιασμένων Κλάσεων

- Για την δημιουργία/αρχικοποίηση κάποιου αντικειμένου μίας φωλιασμένης κλάσης χρειαζόμαστε αντικείμενο της εξωτερικής κλάσης
- Σύνταξη:

```
<outer class>.<nested class> =
        <outer class object>.new <nested class>();
```

- Παράδειγμα 1 (πρώτα δημιουργία αντικείμενου υπερκλάσης):

```
1.  NestedClassExample o = new NestedClassExample();
2.  NestedClassExample.OKListenerClass oko = o.new
OKListenerClass();
```

- Παράδειγμα 2 (ταυτόχρονη δημιουργία αντικείμενου υπερκλάσης):

```
1. NestedClassExample.OKListenerClass oko = (new
NestedClassExample()).new OKListenerClass();
```

# Φωλιασμένες Κλάσεις – Ενθυλάκωση (συν.)

- Όπως συμβαίνει για όλα τα **μέλη** μιας κλάσης, έτσι και για τις εσωτερικές κλάσεις μπορούμε να καθορίζουμε την εμβέλεια τους σαν **public**, **private**, **protected** ή **friendly**.
- Με τον ορισμό τους σαν **private** μπορούμε να αναπαραστήσουμε πλήρως την σχέση της **σύνθεσης**


- Επίσης οι **φωλιασμένες κλάσεις** μπορούν να οριστούν σαν στατικές **(static)**
- Αυτό σημαίνει ότι μπορούμε να δημιουργήσουμε αντικείμενα της φωλιασμένης κλάσης χωρίς να δημιουργήσουμε αντικείμενα της εξωτερικής κλάσης
- Παράδειγμα: NestedClassExample.OKListenerClass oko3 = **new NestedClassExample.OKListenerClass();**

# Εσωτερικές Κλάσεις (Inner Classes)

- Οι **φωλιασμένες μη στατικές κλάσεις** ονομάζονται **εσωτερικές κλάσεις (inner classes)**

- Μία **εσωτερική κλάση σχετίζεται πάντα** με ένα **στιγμιότυπο της περικλείουσας εξωτερικής κλάσης** όπως οι μεταβλητές και οι συναρτήσεις της κλάσης
- Για το λόγο αυτό **δεν μπορεί να δηλώσει κανένα στατικό μέλος**

- Ένα **στιγμιότυπο εσωτερικής κλάσης υπάρχει/ζει μόνο μέσα στα πλαίσια ενός στιγμιότυπου της εξωτερικής κλάσης**

# Εσωτερικές κλάσεις και upcasting

- 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 διεπαφή…
  - **.. να μπορούμε να κάνουμε αναβάθμιση αντικειμένων (upcasting) στη διεπαφή αυτή, αποκρύπτοντας ταυτόχρονα την υλοποίηση της κλάσης του αντικειμένου το οποίο αναβαθμίζουμε.**
- Μία διεπαφής περιέχει κατ’ ανάγκη **δημόσιες** μεθόδους. Επομένως, δεν είναι δυνατή η απόκρυψη των δημόσιων μεθόδων.

# Εσωτερικές Κλάσεις: Απόκρυψη Πληροφορίας

```
interface IDestination { String readLabel(); }
interface IInner { int value(); }

class OuterClass {
  private class PrivateInnerClass implements IInner {
    private int i = 11;
    public int value() { return i; }
  }
  class FriendlyInnerClass implements IDestination {
    private String label;
    private FriendlyInnerClass(String s) { label = s; }
    public String readLabel() { return label; }
    public void invinsible(){}
  }

  public IDestination dest(String s) { return new
FriendlyInnerClass(s);  }
  public IInner cont() { return new PrivateInnerClass(); }
}
public class TestOuter {
  public static void main(String[] args) {
    OuterClass p = new OuterClass();
    IInner c = p.cont();
    IDestination d = p.dest("Tanzania");
    d.invinsible();

    Outer3.PrivateInnerClass pc = p.new PrivateInnerClass();
    }
}
```

**IDestination δεν έχει την μέθοδο invinsible**

**Η PrivateInnerClass είναι private**

# Εσωτερικές «Τοπικές» και «Ανώνυμες» 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 διεπαφή (interface).
- Σαν ανώνυμες κλάσεις που επεκτείνουν μια κλάση με μη προκαθορισμένο κατασκευαστή.
- κ.ο.κ.

# Εσωτερικές «Τοπικές» Κλάσεις

- Οι εσωτερικές «τοπικές» κλάσεις δηλώνονται μέσα σε μία μέθοδο ή κατασκευαστή
- Η εμβέλεια τους καθορίζεται ανάλογα με του που έχουν οριστεί
- Μπορούν να καλέσουν όποιο κατασκευαστή της υπερκλάσης έχουν πρόσβαση
- Χρησιμοποιούνται συνήθως για απόκρυψη πληροφοριών με την υλοποίηση μίας διαπροσωπείας

# Παράδειγμα Εσωτερικής «Τοπικής» κλάσης

```
public class InnerLocalClass  extends JFrame {

  JButton jbtOK = new JButton("OK");

  public InnerLocalClass() {
setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
    add(jbtOK);
    setActionListener();
  }
  private void setActionListener(){
    class OKListenerClass implements ActionListener {
      public void actionPerformed(ActionEvent e) {
        System.out.println("OK");
      }
    }
    jbtOK.addActionListener(new OKListenerClass());
  }

  public static void main(String[] args) {
    JFrame frame = new InnerLocalClass();
    frame.setTitle("InnerLocalClass");
    …
  }
}
```

**Εσωτερική Τοπική Κλάση ορισμένη στη μέθοδο `setActionListener()`**

# Παράδειγμα 2 Εσωτερικής «Τοπικής» κλάσης

```
interface IInnerLocal {int getX(); }

public class InnerLocal {
  int x;
  int y;

  void setX(int x){this.x = x;}
  int getX() {return this.x;}

  void setY(int y){this.y = y;}
  int getY() {return this.y;}

  public IInnerLocal getObject(){

    class ILocalX implements IInnerLocal {
      private int x;
      ILocalX(int x){this.x = x;}
      public int getX() { return this.x; }
    }

    return new ILocalX(x);
  }
}
```

Επιστροφή Τύπου Διαπροσωπείας αφού η κλάση του αντικειμένου που θα επιστρέψουμε δεν έχει ακόμα οριστεί.

Εσωτερική Τοπική Κλάση ορισμένη στη μέθοδο getObjecT()

Επιστρέφουμε καινούριο αντικείμενο από την κλάση που μόλις ορίσαμε

# Εμβέλεια Εσωτερικής «Τοπικής» κλάσης

```
interface IInnerLocal {int getX(); }

public class InnerLocal {
  …

  public IInnerLocal getObject(int y){
    if(y>0) {
      class ILocalX implements IInnerLocal {
        private int x;
        ILocalX(int x){this.x = x;}
        public int getX() { return this.x; }
      }
      return new ILocalX(x);
    }
    else {
      return new ILocalX(y);
    }

  }
}
```

Εσωτερική Τοπική Κλάση ορισμένη μέσα στην εμβέλεια του `if` …

Δεν ισχύει στην εμβέλεια του `else`.

# Εσωτερικές «Ανώνυμες» Κλάσεις

- Μία **εσωτερική «ανώνυμη» κλάση** είναι μία **κλάση χωρίς όνομα**
- **Συνδυάζουν τον ορισμό και δημιουργία στιγμιότυπου μίας εσωτερικής κλάσης σε ένα βήμα**
- Σύνταξη: (μέσα σε μία μέθοδο ή κατασκευαστή)

```
new SuperClassName/InterfaceName() {
  //1. Υλοποίηση ή Υπερσκέλιση μεθόδων
  //   στην υπερκλάση ή διαπροσωπεία
  //2. Υλοποίηση άλλων απαραίτητων μεθόδων
  //   που μπορεί να χρειαστούν
}
```

# Εσωτερικές «Ανώνυμες» Κλάσεις (συν.)

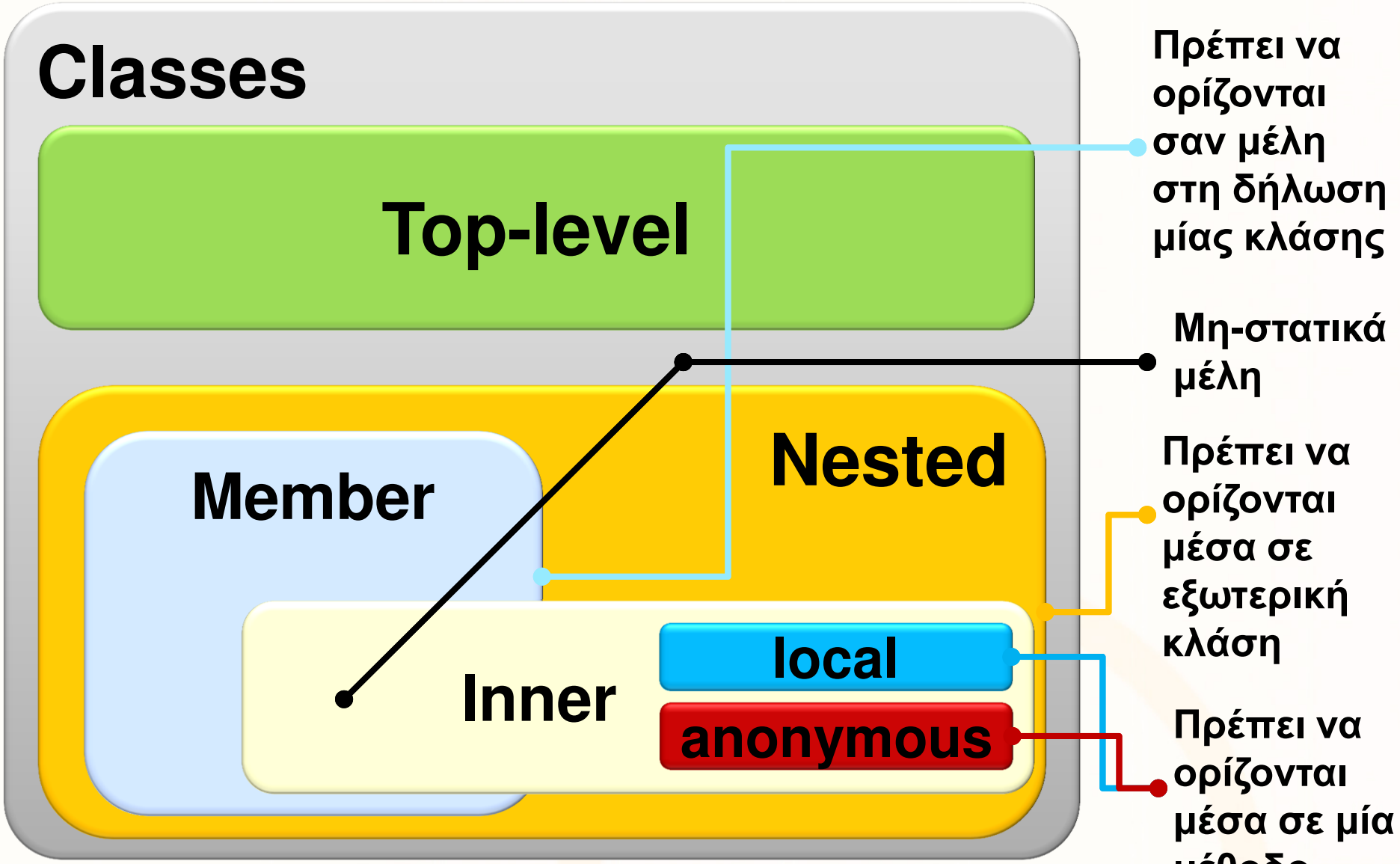
- Οι **εσωτερικές «ανώνυμες» κλάσεις πρέπει πάντα έμμεσα** να κληρονομούν από μία υπερκλάση ή να υλοποιούν μία διαπροσωπεία
- Δεν μπορούν να δηλώσουν άμεσα (δηλ. extends ή implements)
- Μία εσωτερική «ανώνυμη» κλάση πρέπει να υλοποιήσει όλες τις abstract μεθόδους της υπερκλάσης ή διαπροσωπείας
- Πάντα χρησιμοποιούν τον default no-arg constructor της υπερκλάσης. Αν υλοποιούν διαπροσωπεία τότε καλείται ο κατασκευαστής της Object().

# Παράδειγμα Εσωτερικής «Ανώνυμης» κλάσης

```
public class InnerAnonymousClass  extends JFrame {
  JButton jbtOK = new JButton("OK");

  public InnerAnonymousClass() {
setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
    add(jbtOK);

    jbtOK.addActionListener(
       new ActionListener(){
         public void actionPerformed(ActionEvent e) {
           System.out.println("OK");
         }
       }
    );
  }

  public static void main(String[] args) {
    InnerAnonymousClass frame = new
                                InnerAnonymousClass();
    frame.setTitle("InnerAnonymousClass");
    …
  }
}
```

**Εσωτερική Ανώνυμη Κλάση ορισμένη σαν παράμετρος στη μέθοδο addActionListener**

# Σύνοψη – Ταξινομία Κλάσεων




https://blogs.oracle.com/darcy/entry/nested_inner_member_and_top

# Φωλιασμένες Κλάσεις - Αποθήκευση σε Αρχεία

- Ο μεταγλωττιστής παράγει .class αρχεία για όλες τις κλάσεις που χρησιμοποιούνται
- Για top-level κλάσεις παράγονται αρχεία με το ίδιο όνομα με τα java αρχεία που είναι αποθηκευμένα
- Τι γίνεται στην περίπτωση των φωλιασμένων κλάσεων;
  - **Φωλιασμένες και Κλάσεις Μέλη** ονομάζονται με το όνομα της top-level κλάσης, $ και μετά το δικό τους όνομα.
    Π.χ., **<outer class>$<nested class>**.class
  - Στις **Εσωτερικές Τοπικές Κλάσεις**, μετά το $, υπάρχει κάποιος αύξων αριθμός (δηλ., 1,2,...) και μετά το δικό τους όνομα.
    Π.χ., **<outer class>$1<nested class>**.class
  - Στις **Εσωτερικές Ανώνυμες Κλάσεις**, μετά το $, υπάρχει μόνο κάποιος αύξων αριθμός (δηλ., 1,2,...)
    Π.χ., **<outer class>$1**.class

# Φωλιασμένες Κλάσεις-Αποθήκευση σε Αρχεία (συν.)

```
class OuterClass {

  static class NestedClass {}

  class InnerMemberClass {}

  OuterClass(){
    class InnerLocalClass { int x;}
  }

  private Object method() {
    return new Object() { int x=5; };
  }

  public static void main(
                          String[] args) {
    OuterClass o = new OuterClass();
  }
}
```

OuterClass.class

OuterClass$NestedClass.class

OuterClass$InnerMemberClass.class

OuterClass$1InnerLocalClass.class

OuterClass$1.class

# Εκτύπωση Χαρακτηριστικών Κλάσης

- Κάθε αντικείμενο (και υποκλάσεις) περιέχει την μέθοδο getClass() που επιστρέφει χρήσιμα χαρακτηριστικά για το αντικείμενο
- Παράδειγμα Χρήσης της getClass()

```
public static void print(Object o){
 System.out.println("Class=" + o.getClass());
 System.out.println("Enclosing Class=" +
                                    o.getClass().getEnclosingClass());
 System.out.println("Enclosing Constructor=" +
                                  o.getClass().getEnclosingConstructor());
 System.out.println("Enclosing Method=" +
                                       o.getClass().getEnclosingMethod());
 System.out.println("Is Member Class?=" +
                                         o.getClass().isMemberClass());
 System.out.println("Is Local Class?=" + o.getClass().isLocalClass());
 System.out.println("Is Anonymous Class?=" +
                                        o.getClass().isAnonymousClass());
 System.out.println(); }
```

# Παράδειγμα Χρήσης

```
interface ITest { public void test(); }
interface IOuter { void anonymous(ITest t); }
class OuterClass{
  static class NestedClass
{NestedClass(){print(this);}}
  class InnerMemberClass{
    InnerMemberClass(){ print(this);} }
  OuterClass(){
    print(this);
    new ITest(){
      public void test(){
        print(this);
      }}.test();
  }
  public void local() {
    class InnerLocalClass {
      InnerLocalClass(){print(this);} }
      InnerLocalClass il = new
InnerLocalClass();
    }
}

public class NestedFileSystem {
  public static void main(String[] args) {
    OuterClass o = new OuterClass();
    OuterClass.NestedClass n =
                  new OuterClass.NestedClass();
    OuterClass.InnerMemberClass im = (new
         OuterClass()).new InnerMemberClass();
    o.local();
  }
}
```

```
Class=class OuterClass
Enclosing Class=null
Enclosing Constructor=null
Enclosing Method=null
Is Member Class?=false
Is Local Class?=false
Is Anonymous Class?=false

Class=class OuterClass$1
Enclosing Class=class OuterClass
Enclosing
Constructor=OuterClass()
Enclosing Method=null
Is Member Class?=false
Is Local Class?=false
Is Anonymous Class?=true

Class=class
OuterClass$NestedClass
Enclosing Class=class OuterClass
Enclosing Constructor=null
Enclosing Method=null
Is Member Class?=true
Is Local Class?=false
Is Anonymous Class?=false
```

```
Class=class OuterClass$1
Enclosing Class=class OuterClass
Enclosing
Constructor=OuterClass()
Enclosing Method=null
Is Member Class?=false
Is Local Class?=false
Is Anonymous Class?=true

Class=class
OuterClass$InnerMemberClass
Enclosing Class=class OuterClass
Enclosing Constructor=null
Enclosing Method=null
Is Member Class?=true
Is Local Class?=false
Is Anonymous Class?=false

Class=class
OuterClass$1InnerLocalClass
Enclosing Class=class OuterClass
Enclosing Constructor=null
Enclosing Method=public void
OuterClass.local()
Is Member Class?=false
Is Local Class?=true
Is Anonymous Class?=false
```