Προγραμματισμός ΙΙ (Java)

11. Προχωρημένα θέματα Προγραμματισμού με αντικείμενα

Java Enterprise Technologies

Εσωτερικές κλάσεις

Εσωτερικά interfaces και κλάσεις

- Μέσα σε ένα interface (σε μία κλάση) μπορούμε να δηλώσουμε ένα άλλο interface (μια κλάση)
- Χρησιμοποιούνται
  - για να ομαδοποιήσουμε σχετικά μεταξύ τους interfaces ή κλάσεις (λειτουργικότητα), που δε χρησιμοποιούνται σε άλλο περιεχόμενο.
  - για να διαχωρίσουμε τη λειτουργικότητα σε μια κλάση από την ίδια την κλάση
- Παράδειγμα:
  - μια κλάση BinarySearchTree μπορεί να έχει εκτός από τις μεθόδους για προσθήκη και αφαίρεση κόμβων, και την εσωτερική κλάση BinarySearchTreeNode
  - Μέθοδοι που υλοποιούν κάποιο αλγόριθμο ταξινόμησης ή αναζήτησης κόμβων ομαδοποιούνται σε εσωτερική κλάση της BinarySearchTree. Για παράδειγμα ένας Iterator για τη διάσχιση του δέντρου.

Εσωτερικές (inner) κλάσεις

- Έχουν πρόσβαση στα μέλη της κλάσης που τις περιέχει
- Η εξωτερική κλάση δεν μπορεί να προσπελάσει private μέλη της εσωτερικής
- Δεν είναι το ίδιο πράγμα με τη σύνθεση
- Επιτρέπουν την καλύτερη οργάνωση των κλάσεων
- Είναι τεσσάρων τύπων
  - Static member classes
  - Member classes
  - Local classes
  - Anonymous classes

Static inner κλάσεις

- Στατικό μέλος μιας κλάσης όπως οι μέθοδοι.
- Έχει πρόσβαση ***μόνο*** στα στατικά μέλη της κλάσης που την περιέχει
- Μπορεί να περιέχει δικά της στατικά μέλη
- Μπορούμε να φτιάξουμε στιγμιότυπα static inner κλάσεων που όμως δε σχετίζονται με το εξωτερικό αντικείμενο (ανήκουν στην κλάση)
- Φωλιασμένες κλάσεις χωρίς μεγάλη χρησιμότητα

```
class BinarySearchTree {
   public static class BSTNode {
      BSTNode left;
      BSTNode right;
      Human value;
      public BSTNode(Human value) {   this.value = value;     }
   }
   protected BSTNode root;

   public BinarySearchTree() {      root = null;    }
   public void insert(Human x) {        root = insert(x, root);    }
   public void remove(Human x) {       root = remove(x, root);    }
   public BSTNode find(Human x) {        return find(x, root);    }
   protected BSTNode insert(Human x, BSTNode t) {
                if (t == null) {           t = new BSTNode(x);        }
     else if (x.compareTo(t.value) < 0) {
                 t.left = insert(x, t.left);      }
     else if (x.compareTo(t.value) > 0) {
                t.right = insert(x, t.right);    }
     else {    System.out.println(x
                + " is already in the tree");      }
      return t;
   }
...
}
```

```
public static void main(String args[]){
Human a1=new Human(1,"joe","black");
Human a2=new Human(2,"bill","brows");
Human a3=new Human(3,"mary","pierce");
BinarySearchTree bst=new BinarySearchTree();
bst.insert(a1);
bst.insert(a2);
bst.insert(a3);
bst.insert(a1);
}
```

Παράδειγμα

```
class Parcel{
    boolean local;
public static class CostHandler{
    private int cost=10;
    public int getCost() { return cost; }
    public void setCost(int c) {cost=c;}
}
}
```

- Η μεταγλώττιση παράγει τα αρχεία Parcel και Parcel$CostHandler.class

```
Parcel.CostHandler p1=new Parcel.CostHandler();
Parcel.CostHandler p2 = new Parcel.CostHandler();
p1.setCost(20);
System.out.println(“p1:"+p1.getCost()+” p2:”+ p2.getCost());
```

Non-static inner κλάσεις

- Επώνυμες κλάσεις μέλη: Ανήκουν στο κάθε αντικείμενο της κλάσης και έχουν πρόσβαση σε όλα τα μέλη του
- Τοπικές κλάσεις: Δηλώνονται σε ένα block κώδικα και είναι ορατές μόνο σε αυτό
- Ανώνυμες κλάσεις: Είναι τοπικές κλάσεις χωρίς όνομα.
- Παράδειγμα: Έστω ότι υπάρχει μια κλάση SearchCriteria με συγκεκριμένη λειτουργικότητα. Η ακόλουθη μέθοδος search:

```
myOrder.search( new SearchCriteria () {
                public boolean matches(Object o) { ... }
                }
        ) ;
```

  - Δημιουργεί μια ανώνυμη κλάση που επεκτείνει την SearchCriteria. Έχει μόνο μια μέθοδο, αλλά μπορούμε να ορίσουμε μεταβλητές και άλλες μεθόδους αν θέλουμε.

```
class BinarySearchTree {
   public static class BSTNode {
      BSTNode left;
      BSTNode right;
      Human value;
      public BSTNode(Human value) {          this.value = value;       }
   }

   public class InOrderIterator{
      public void iterateInOrder(){       inorder(root);       }
      public void iteratePreOrder(){          preorder(root);       }
      public void iteratePostOrder(){          postorder(root);       }
      private void inorder(BSTNode t){
        if (t.left  != null )        inorder(t.left);
        System.out.println(t.value);
        if (t.right != null)        inorder(t.right);
      }
     …
   }

   protected BSTNode root;
…
}
```

```
public static void main(String args[]){
Human a1=new Human(1,"joe","black");
Human a2=new Human(2,"bill","brows");
Human a3=new Human(3,"mary","pierce");
BinarySearchTree bst=new BinarySearchTree();
bst.insert(a3);
bst.insert(a1);
bst.insert(a2);
BinarySearchTree.InOrderIterator it =
bst.new InOrderIterator();
it.iterateInOrder();
}
```

Παράδειγμα

```
class Parcel{
    boolean isLocal; private int cost=10;
public class CostHandler{
        public int getCost() {
                if (isLocal) cost=2*cost;
                return cost;
    }}}
```

- Χρειαζόμαστε πλέον ένα αντικείμενο Parcel

```
Parcel p=new Parcel();
p.isLocal=true;
Parcel.CostHandler p1 = p.new CostHandler();
System.out.println(“p1:"+p1.getCost());
```

Πλεονεκτήματα

- Αντικειμενοστραφής κώδικας
  - Κάθε αντικείμενο της εσωτερικής κλάσης έχει πρόσβαση στα μέλη της εξωτερικής κλάσης
  - Ταυτόχρονα είναι ένα ξεχωριστό αντικείμενο.
  - Ομαδοποίηση της συμπεριφοράς

    π.χ. Ένας αλγόριθμος διάσχισης σε ένα ArrayList μπορεί να διαχωριστεί από τις μεθόδους κατασκευής του ArrayList με χρήση μιας εσωτερικής κλάσης (iterator).
- Οργάνωση κώδικα
  - Ιεραρχίες εσωτερικών κλάσεων
- Κλήσεις στο εξωτερικό αντικείμενο (callback)

Μειονεκτήματα των inner κλάσεων

- Δύσκολες στην κατανόησή τους. Απαιτούν εμπειρία.
- Αυξάνει το συνολικό αριθμό κλάσεων στον κώδικα, άρα και την πολυπλοκότητα
- Τα περισσότερα εργαλεία ανάπτυξης προσφέρουν περιορισμένη υποστήριξη
  - Δεν έχουν δυνατότητα επισκόπησης των μελών της εσωτερικής κλάσης

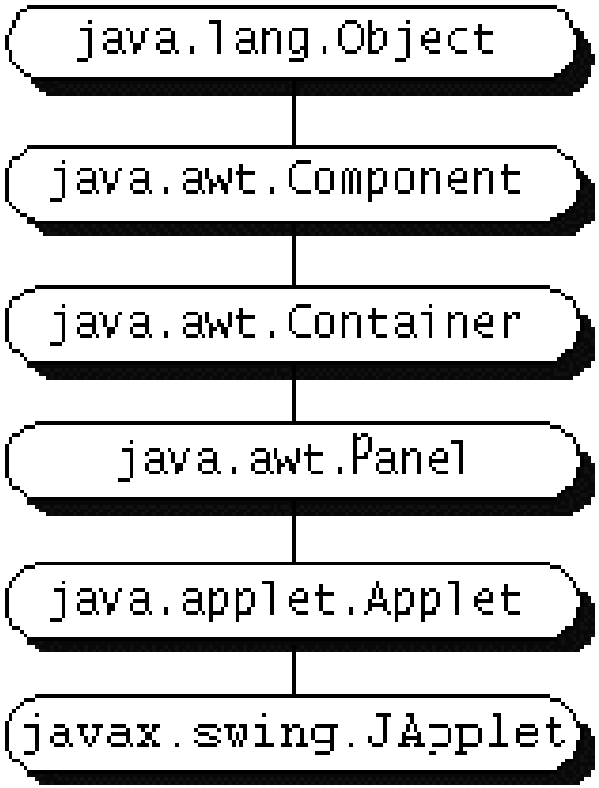
Java applets

Τι είναι το Applet

- Είναι ένα πρόγραμμα Java που τρέχει στο περιβάλλον μιας εφαρμογής (συνήθως σε ένα φυλλομετρητή – browser ή στο πρόγραμμα appletviewer του jdk).
- Η εφαρμογή διαθέτει δικό της JVM στο οποίο τρέχει το applet.
- Το applet προσφέρεται μέσα από ένα HTML έγγραφο. Στην ετικέτα APPLET προσδιορίζεται το URL του applet.
- Κάθε applet επεκτείνει την τάξη java.applet.Applet ή την javax.swing.JApplet που επίσης επεκτείνει την Applet.
- Για λόγους ασφαλείας τα applets τρέχουν με περιορισμούς (in a sandbox).

Ιεραρχία τάξεων

- Ένα applet είναι στιγμιότυπο μιας απογόνου των `Applet` ή `JApplet`.
- Είναι ένα `Panel`, και συνεπώς ένα παράθυρο `Container`.
- Μια τάξη applet **πρέπει να είναι public**.
- Κάθε applet κληρονομεί τις μεθόδους `init`, `start`, `paint`, `stop`, και `destroy`.



Τι κάνουν οι μέθοδοι που κληρονομεί

- init(): καλείται μόλις φορτώνεται το applet και το αρχικοποιεί
- start(): καλείται όποτε ο browser έρθει στο προσκήνιο (focus) και εκτελεί τις λειτουργίες του applet
- stop(): καλείται όταν ο browser ελαχιστοποιείται ή όταν ο χρήστης φεύγει από μια σελίδα
- destroy(): καλείται όταν κλείσει ο φυλλομετρητής και κάνει ότι χρειάζεται πριν τερματίσει το applet

Στα applet δεν υπάρχει main (αυτή την παρέχει ο φυλλομετρητής)

Υπέρβαση μεθόδων

- Αν δεν υπερβούμε τις μεθόδους αυτές το applet δεν κάνει τίποτε
  - init(): Καλείται μία φορά. Περιέχει τον κώδικα που κατασκευάζει ότι χρειαζόμαστε (αρχικοποιεί μεταβλητές, GUI, ακροατές κλπ).
  - start(): Περιέχει το σώμα εντολών του applet που θέλουμε να ξανατρέχει κάθε φορά που εμφανίζεται το applet.
  - stop(): Πρέπει να σταματά την εκτέλεση του applet (π.χ. Ένα χρονόμετρο). Χρησιμοποιείται σε συνδυασμό με τη start για να διακόπτει χρονοβόρες διεργασίες όταν το applet είναι στο φόντο.
  - destroy(): Ελευθερώνει πόρους του συστήματος (π.χ. Κλείνει συνδέσεις σε ΒΔ κλπ.).

΄Ενα απλό applet

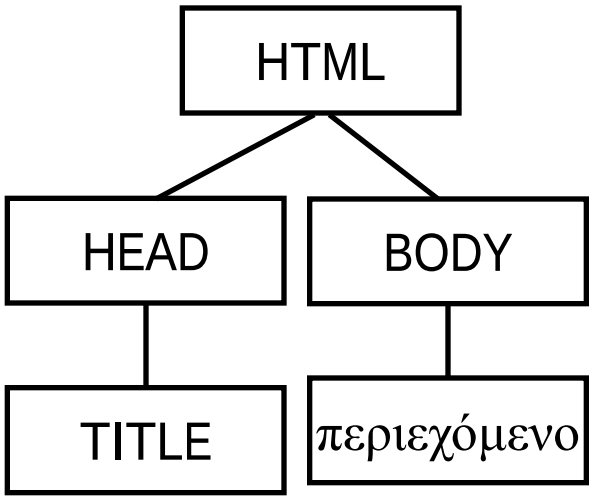
```
import javax.swing.*;
import java.awt.*;
public class HiWorld extends JApplet {
    public void init() {
      getContentPane().add(
                        new JLabel(“Hi World!"));
    }
}
```

Δομή μιας σελίδας HTML

```
<html>
  <head>
     <title> Hi World Applet </title>
  </head>

  <body>
        <applet code="HiWorld.class”
          width=300 height=200>
        </applet>
  </body>
</html>
```

Test.html



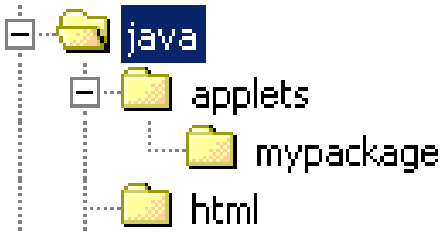
Εκτέλεση από command line

- appletviewer c:\java\Test.html
- Στην περίπτωση αυτή το HiWorld.class βρίσκεται στο c:\java\

Οργάνωση αρχείων

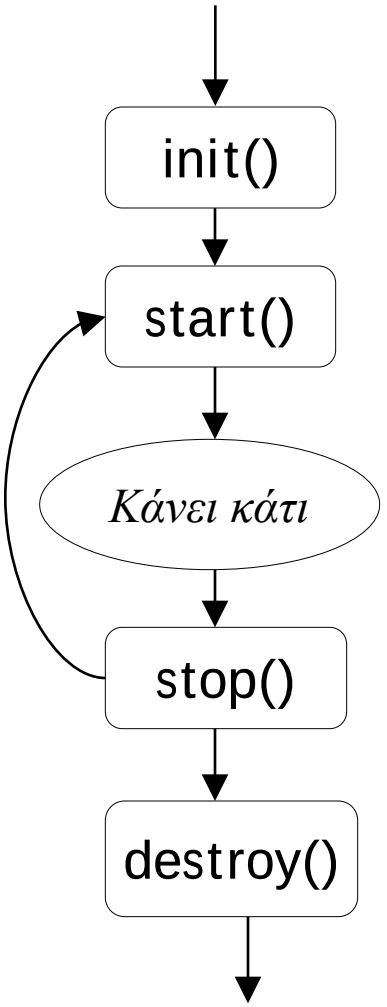
- Αν το applet μας δεν είναι σε κάποιο package αρκεί το html και το class να είναι στο ίδιο φάκελο
- Διαφορετικά πρέπει η οργάνωση των φακέλων να ακολουθήσει αυτή των packages
- Αν τα applets βρίσκονται σε άλλο φάκελο από αυτό του html τότε χρησιμοποιούμε την παράμετρο codebase.

```
<applet codebase="../applets"
  code=mypackage/AppletClassName.class
  width=100 height=200> </applet>
```



Κύκλος ζωής ενός Applet

- Μόλις ο browser εντοπίσει μια ετικέτα <APPLET>
  - Δεσμεύει το χώρο που καθορίζουν οι παράμετροι width και height για το applet
  - Φορτώνει τα bytecode για τη συγκεκριμένη υποτάξη της Applet
  - Δημιουργεί το αντικείμενο applet
  - Καλεί τη μέθοδο `init` για να αρχικοποιήσει το αντικείμενο (π.χ. Ορίζει χρώμα, γραμματοσειρά κλπ.)
  - Καλεί τη μέθοδο `start`.
  - Στη συνέχεια καλείται η `paint`
    - Αν φύγουμε από τη σελίδα καλείται η μέθοδος `stop`
    - Αν ξαναμπούμε στη σελίδα εκτελείται η μέθοδος `start`.
  - Όποτε κλείσει ο φυλλομετρητής καλείται η μέθοδος `destroy`



Μέθοδοι ενός JApplet (ως Container)

- Container getContentPane()
  - Επιστρέφει το ContentPane του applet
- void setJMenuBar(JMenuBar menuBar)
  - Προσθέτει μενού στο applet
- void setLayout(LayoutManager manager)
  - Ορίζει τη διάταξη στο applet
- System.out.println(String s)
  - Εμφανίζει το μήνυμα σε μια ξεχωριστή κονσόλα

Design patterns

Τι είναι και γιατί τα χρειαζόμαστε

- Συχνά στα προγράμματά μας υπάρχουν επαναλαμβανόμενες ανάγκες για τις οποίες πρέπει να επαναλαμβάνουμε την ίδια λύση με μικρές αλλαγές
- Τα design patterns είναι μια αφηρημένη περιγραφή (σχεδίαση) της λύσης με χρήση αντικειμένων, κλάσεων και διεπαφών και εναλλακτικές υλοποιήσεις ανάλογα με το πρόβλημα.

  a common solution to a recurring problem in design

Στόχοι

- Codify good design
  - Να κάνουμε όσο πιο γενική γίνεται τη σχεδίαση μιας λύσης
- Give design structures explicit names
  - Να δημιουργήσουμε ένα κοινό λεξικό
  - Να μειώσουμε την πολυπλοκότητα
- Capture and preserve design information
  - Να διατηρήσουμε τη δομή του προγράμματος
  - Να βελτιώσουμε την τεκμηρίωση
- Facilitate restructuring/refactoring
  - Να βελτιώσουμε την ανεξαρτησία των λύσεων
  - Να μειώσουμε το χρόνο υλοποίησής τους, χτίζοντας σε προκαθορισμένα σχέδια

Design Pattern Catalogues

- GoF (“the gang of four”) catalogue
  - “Design Patterns: Elements of Reusable Object-Oriented Software,” Gamma, Helm, Johnson, Vlissides, Addison-Wesley, 1995
- POSA catalogue
  - Pattern-Oriented Software Architecture, Buschmann, et al.; Wiley, 1996

GoF Design Patterns

Creational	**Structural**	**Behavioral**
Factory Method Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Flyweight Facade Proxy	Interpreter Template Method Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Παραδείγματα

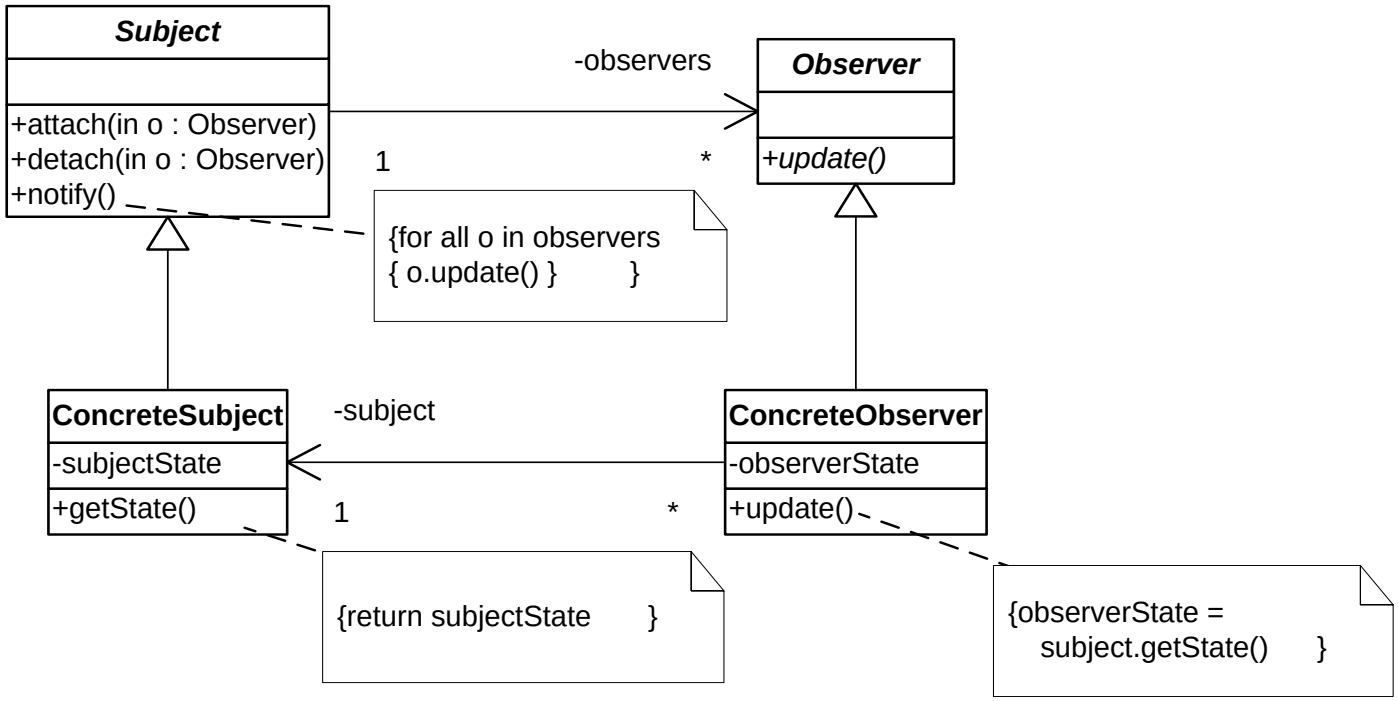
- Observer → MVC
- Singleton

Observer (*Behavioral*)

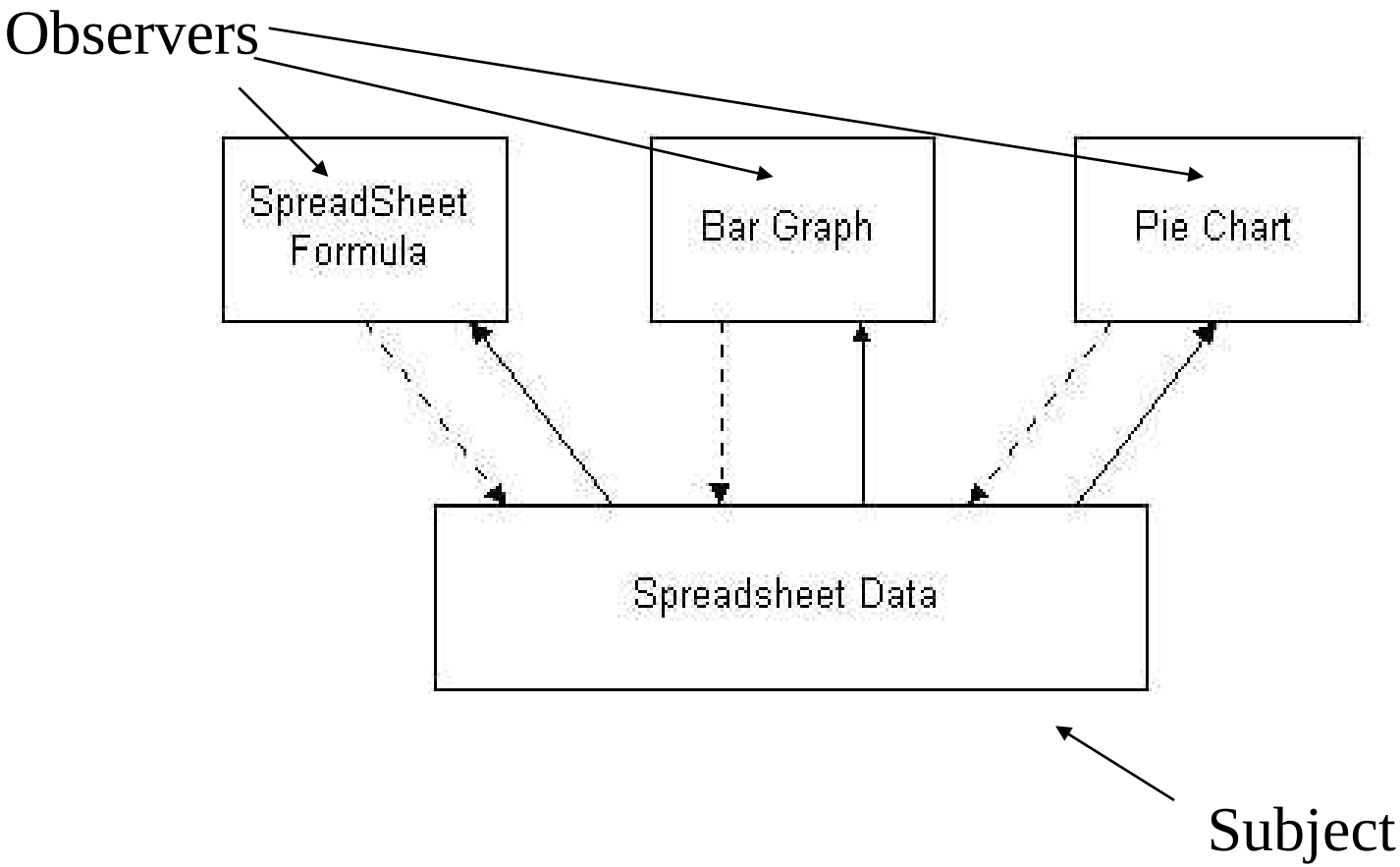
- Σκοπός
  - Μια σχέση εξάρτηση 1:N μεταξύ αντικειμένων. Όταν το 1 αλλάξει ενημερώνει τα υπόλοιπα για να αλλάξουν κι αυτά
- Εφαρμογές
  - Όταν το ίδιο αντικείμενο εμφανίζεται μέσα από δύο ή περισσότερα άλλα αντικείμενα (MVC)
  - Όταν μια αλλαγή στο αντικείμενο, επηρεάζει άλλα, αλλά δεν ξέρουμε ποια τη στιγμή της αλλαγής
  - Όταν ένα αντικείμενο πρέπει να ενημερώσει άλλα αντικείμενα χωρίς να ελέγξει ποια είναι αυτά

Observer (Cont'd)

- Structure



Schematic Observer Example



Observable - Sample Code

```
import java.util.Observable;
public class ObservableValue extends
      Observable
{
   private int n = 0;
   public ObservableValue(int n)
   {
      this.n = n;
   }
   public void setValue(int n)
   {
      this.n = n;
      setChanged();
      notifyObservers();
   }
   public int getValue()
   {
      return n;
   }
}
```

Marks this Observable object as having been changed; the hasChanged method will now return true.

If this object has changed, as indicated by the hasChanged method, then notify all of its observers and then call the clearChanged method to indicate that this object has no longer changed.

Observer - Sample Code

```
import java.util.Observer;
import java.util.Observable;
public class TextObserver implements
      Observer
{
   private ObservableValue ov = null;
   public TextObserver(ObservableValue ov)
   {
      this.ov = ov;
   }
   public void update(Observable obs, Object
       obj)
   {
      if (obs == ov)
      {
         System.out.println(ov.getValue());
      }
   }
}
```

```
public class Main
{
   public Main()
   {
      ObservableValue ov = new ObservableValue(0);
      TextObserver to = new TextObserver(ov);
      ov.addObserver(to);
   }
   public static void main(String [] args)
   {
      Main m = new Main();
   }
}
```

Observer – Sample Code

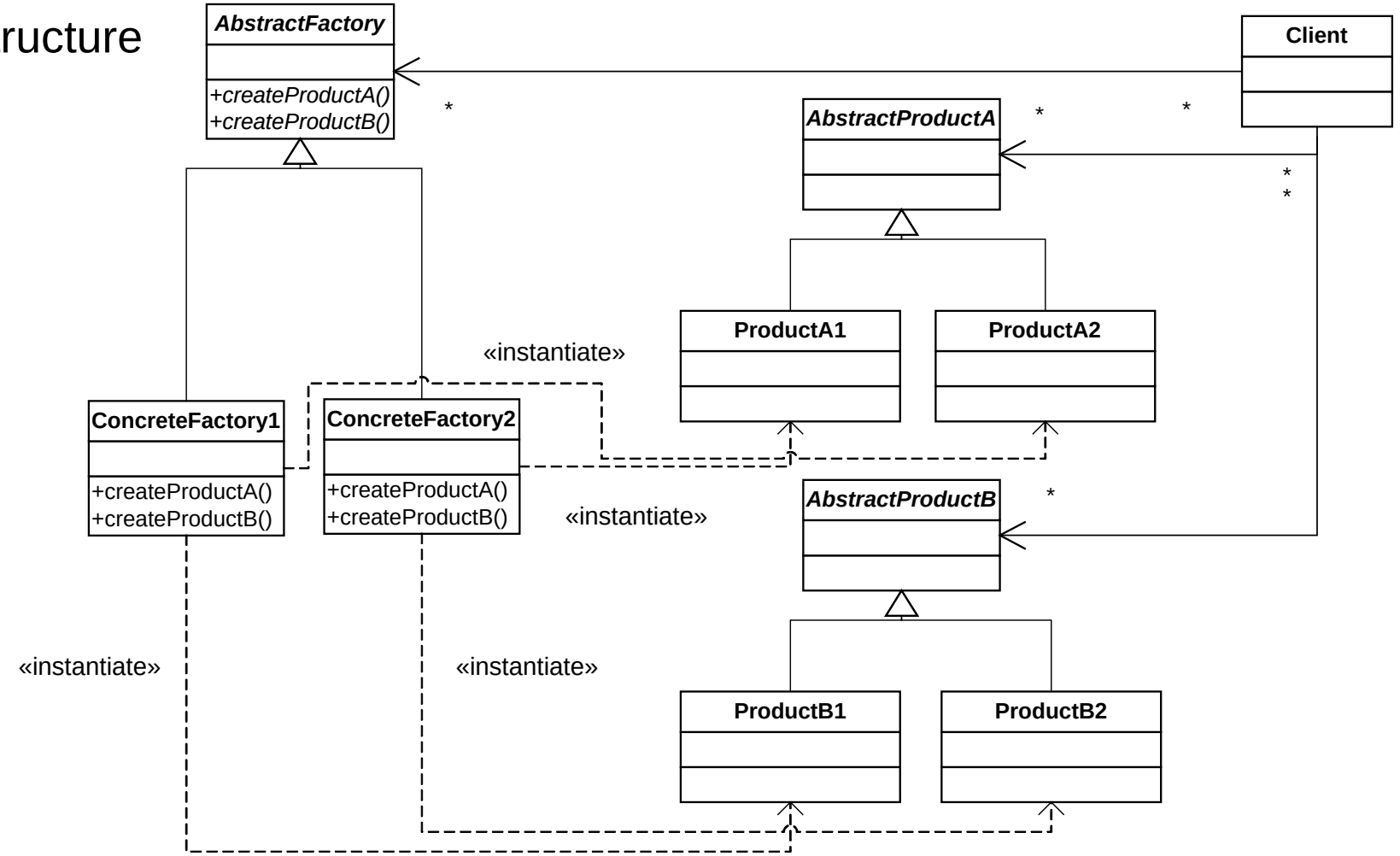
```
public class Main
{
   public Main()
   {
      ObservableValue ov = new ObservableValue(0);
      TextObserver to = new TextObserver(ov);
      ov.addObserver(to);
   }
   public static void main(String [] args)
   {
      Main m = new Main();
   }
}
```

Abstract Factory (*Creational*)

- Σκοπός
  - Να δημιουργεί οικογένειες σχετικών αντικειμένων χωρίς να δίνει το όνομα της κλάσης
- Εφαρμογή
  - Όταν δεν μπορούμε από πριν να προβλέψουμε το αντικείμενο που θέλουμε να δημιουργήσουμε κατά τη χρήση

Abstract Factory

- Structure



Abstract Factory - Example

```
public class MazeFactory {
   public MazeFactory() {...}
   public Maze makeMaze(){
      return new Maze();
  }
  public Room makeRoom(int n) {
       return new Room(n);
  }
  public Wall makeWall() {
       return new Wall();
  }
  public Door makeDoor(Room r1, Room
      r2) {
      return new Door(r1, r2);
  }
};
```

```
public class MazeGame {
// ...
 public Maze createMaze (MazeFactory factory) {
   Maze aMaze = factory.makeMaze();
   Room r1 = factory.makeRoom(1);
   Room r2 = factory.makeRoom(2);
   Door theDoor = factory.makeDoor(r1, r2);
   aMaze.addRoom(r1);
   aMaze.addRoom(r2);
   r1.setSide(MapSite.NORTH, factory.makeWall());
   r1.setSide(MapSite.EAST, theDoor);
   r1.setSide(MapSite.SOUTH, factory.makeWall());
   r1.setSide(MapSite.WEST, factory.makeWall());
   r2.setSide(MapSite.NORTH, factory.makeWall());
   r2.setSide(MapSite.EAST, factory.makeWall());
   r2.setSide(MapSite.SOUTH, factory.makeWall();
   r2.setSide(MapSite.WEST, theDoor); return aMaze;
 }
}
```

Abstract Factory - Example

```
public class EnchantedMazeFactory extends MazeFactory {
      public EnchantedMazeFactory() {...}
       public Room makeRoom(int n) {
            return new EnchantedRoom(n, new Spell());
      }
      public Door makeDoor(Room r1, Room r2) {
          return new DoorNeedingSpell(r1, r2);
      }
}

public class BombedMazeFactory extends MazeFactory {
      public BombedMazeFactory() {...}
     public Wall makeWall(){
         return new BombedWall();
     }
     public Room makeRoom(int n){
        return new RoomWithABomb(n);
     }
}
```

Abstract Factory - Client

```
MazeGame game; // The instance of a game.
…….
Maze aMaze; // A reference to a maze.
switch(choice) {
    case ENCHANTED: {
        EnchantedMazeFactory factory = new EnchantedMazeFactory();
        aMaze = game.createMaze(factory);
        break;
    }
    case BOMBED: {
        BombedMazeFactory factory = new BombedMazeFactory();
        aMaze = game.createMaze(factory);
        break;
    }
}
```

Abstract Factory

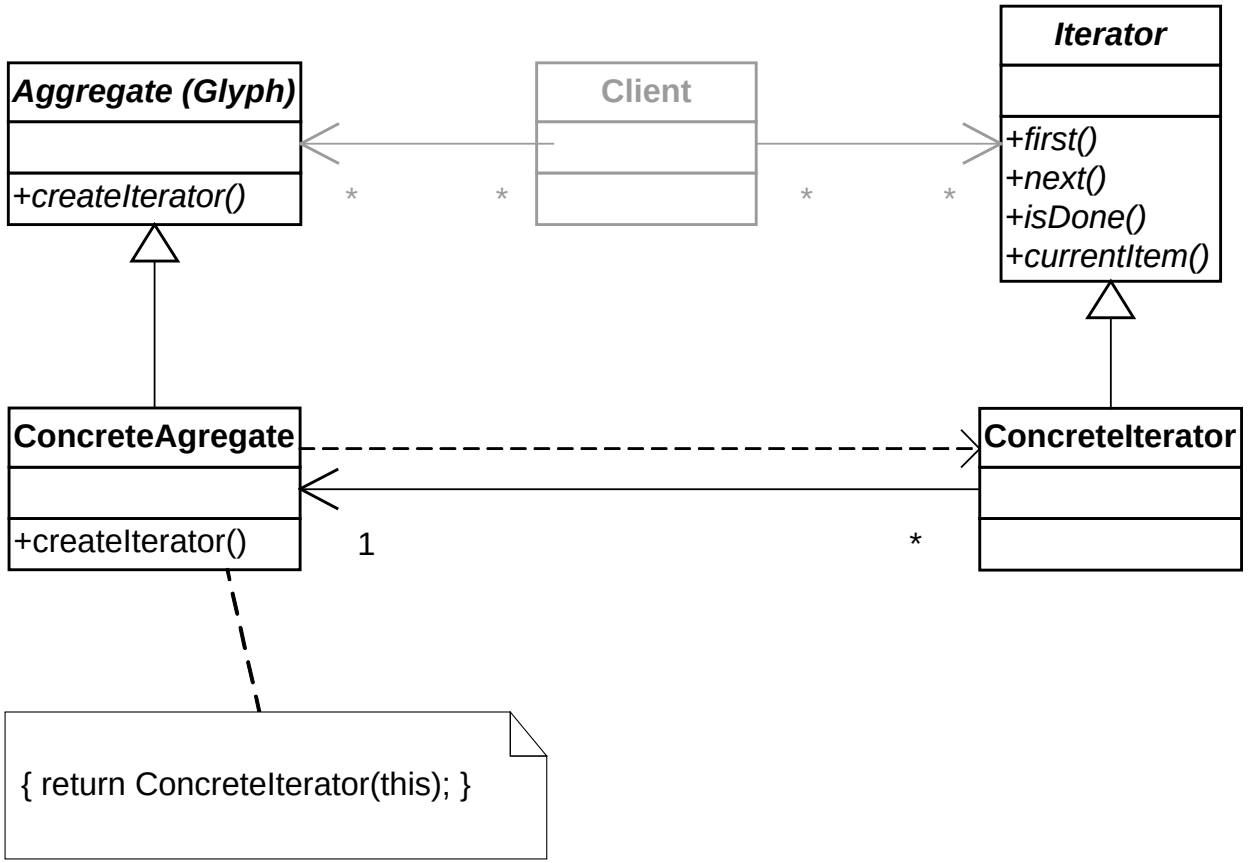
- Πλεονεκτήματα/Μειονεκτήματα
  - + Flexibility: αφαιρεί τα type dependencies κατά τη χρήση
  - + Abstraction: κρύβει τη σύνθεση των αντικειμένων
  - □ Δύσκολο να επεκταθεί το factory interface ώστε να φτιάχνει νέα προϊόντα

Iterator (*Behavioral*)

- Σκοπός
  - Η διάσχιση στοιχείων σειριακά, χωρίς να μας ενδιαφέρει η δομή στην οποία αποθηκεύονται
- Εφαρμογές
  - Αλγόριθμοι διάσχισης
  - Ενιαίος τρόπος διάσχισης σε συλλογές

Iterator

- Structure

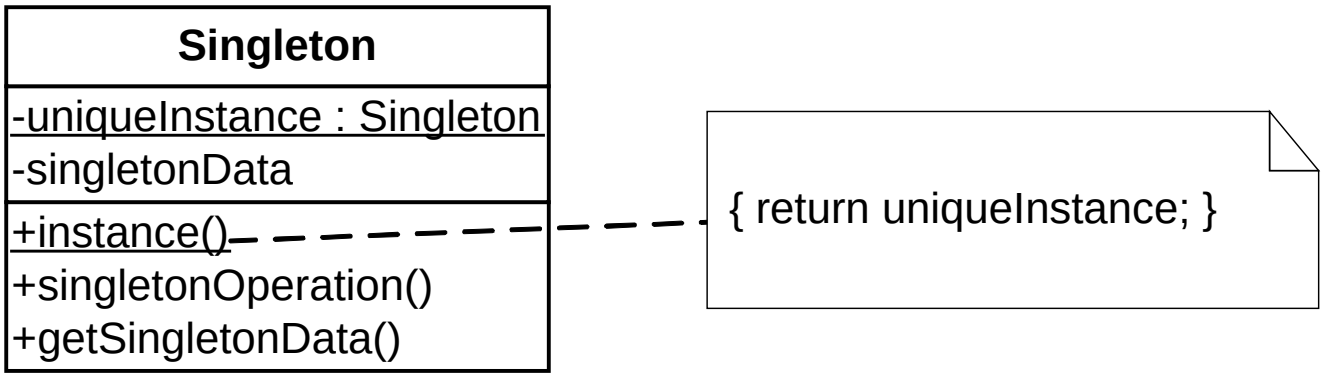


Singleton (*Creational*)

- Σκοπός
  - Να βεβαιωθούμε ότι έχουμε ένα μόνο αντικείμενο από μια κλάση σε όλο το πρόγραμμα και καθολική πρόσβαση από παντού.
- Εφαρμογές
  - Όταν χρειαζόμαστε ακριβώς ένα αντικείμενο από μια κλάση
  - Όταν το μοναδικό αντικείμενο πρέπει να είναι επεκτάσιμο (με κληρονομικότητα) και κατά τη χρήση δεν θέλουμε να τροποποιείται ο κώδικας

Singleton (cont'd)

- Structure



Singleton - Example

```
public class ClassicSingleton {
   private static ClassicSingleton instance = null;
   protected ClassicSingleton() {
      // Exists only to defeat instantiation.
   }
   public static ClassicSingleton getInstance() {
      if(instance == null) {
         instance = new ClassicSingleton();
      }
      return instance;
   }
}
```

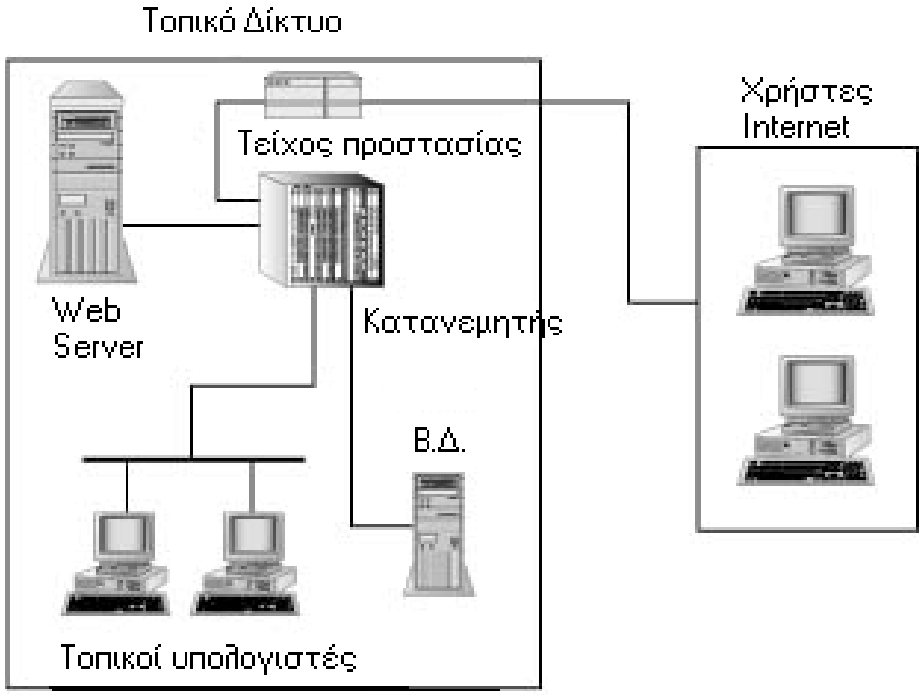
lazy instantiation

```
public class SingletonInstantiator {
  public SingletonInstantiator() {
   ClassicSingleton instance = ClassicSingleton.getInstance();
ClassicSingleton anotherInstance =
new ClassicSingleton();
        ...
  }
}
```

Java Enterprise Technologies

Η αρχιτεκτονική του Web

- Αρχιτεκτονική client/server

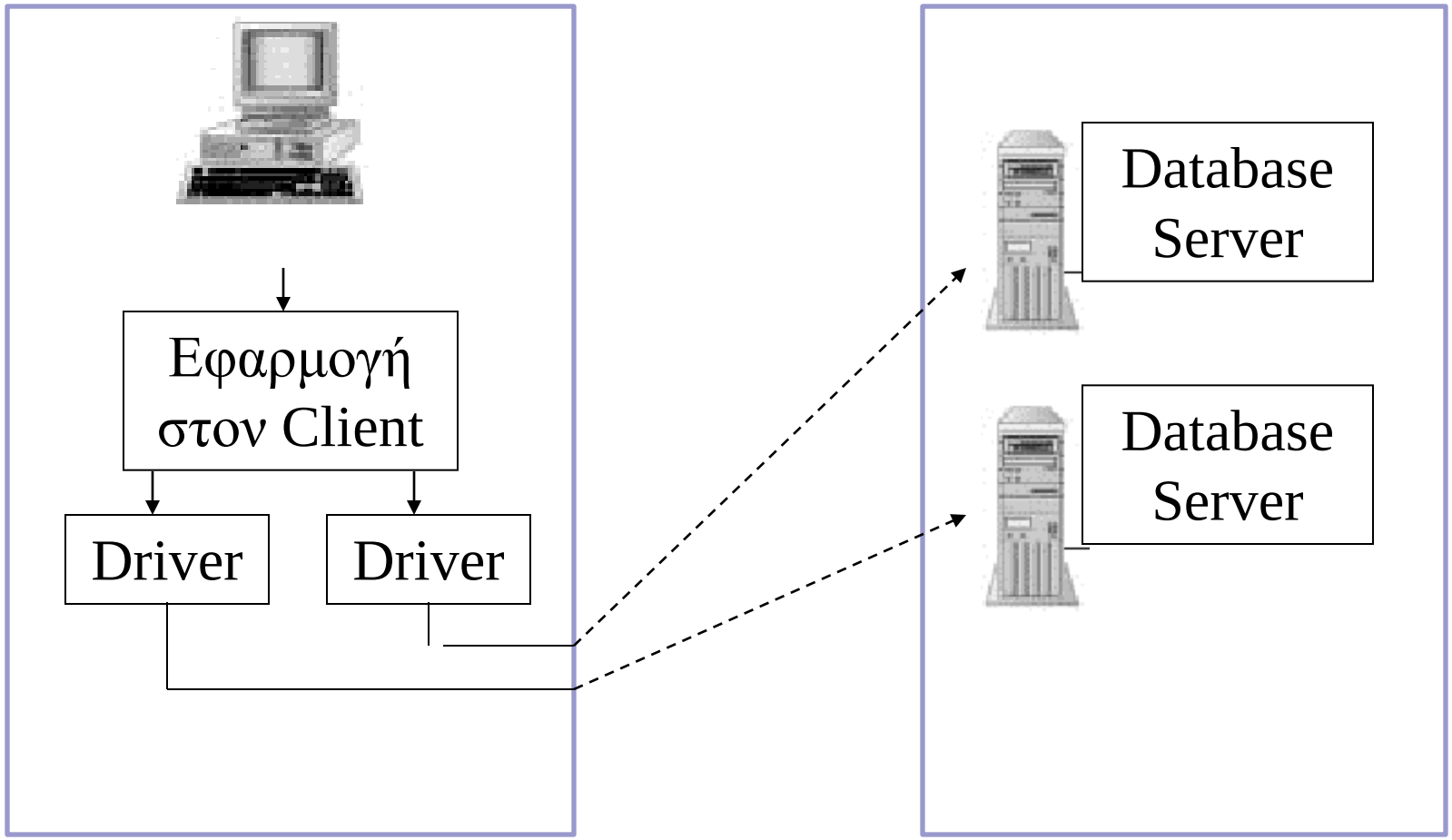


- Αιτήσεις του client - απαντήσεις του server

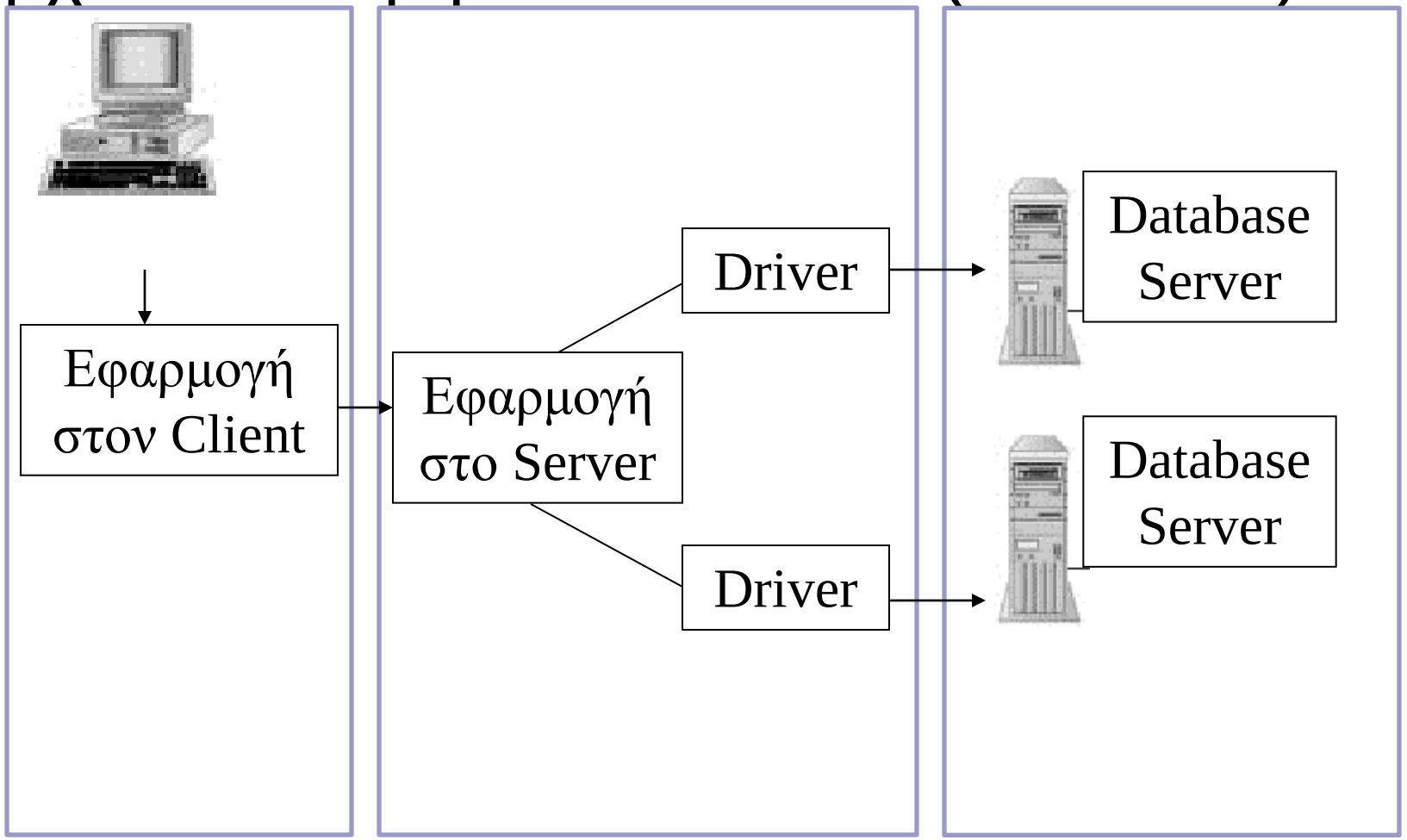
Μορφές αρχιτεκτονικής client-server

- Αρχιτεκτονική δύο επιπέδων (2-tier).
  - Στον client τρέχει μια εφαρμογή που διαβάζει τα περιεχόμενα της ΒΔ στον server.
- Αρχιτεκτονική τριών επιπέδων (3-tier)
  - Η εφαρμογή τρέχει στο server και συνδέει το interface του client με τη ΒΔ.
- Αρχιτεκτονική πολλών επιπέδων (multi-tier)
  - Η εφαρμογή τρέχει διαμοιρασμένη σε περισσότερους servers που επικοινωνούν μεταξύ τους.

Αρχιτεκτονική δύο επιπέδων (Fat client)



Αρχιτεκτονική τριών επιπέδων(thin client)



Απαιτήσεις αρχιτεκτονικής 3-tier

- Ανάπτυξη της διεπαφής με το χρήστη (user-interface)
- Ανάπτυξη της εφαρμογής που θα συνδέει το user-interface με τη βάση δεδομένων (application logic)
- Ανάπτυξη της βάσης δεδομένων (database schema)

Java EE

Java Enterprise Edition

Enterprise Computing

Προκλήσεις

- Portability
- Diverse Environments
- Time-to-market
- Core Competence
- Assembly
- Integration

Τεχνολογίες

- J2SE™
- J2EE™
- JMS
- Servlet
- JSP
- Connector
- XML
- Data Binding
- XSLT

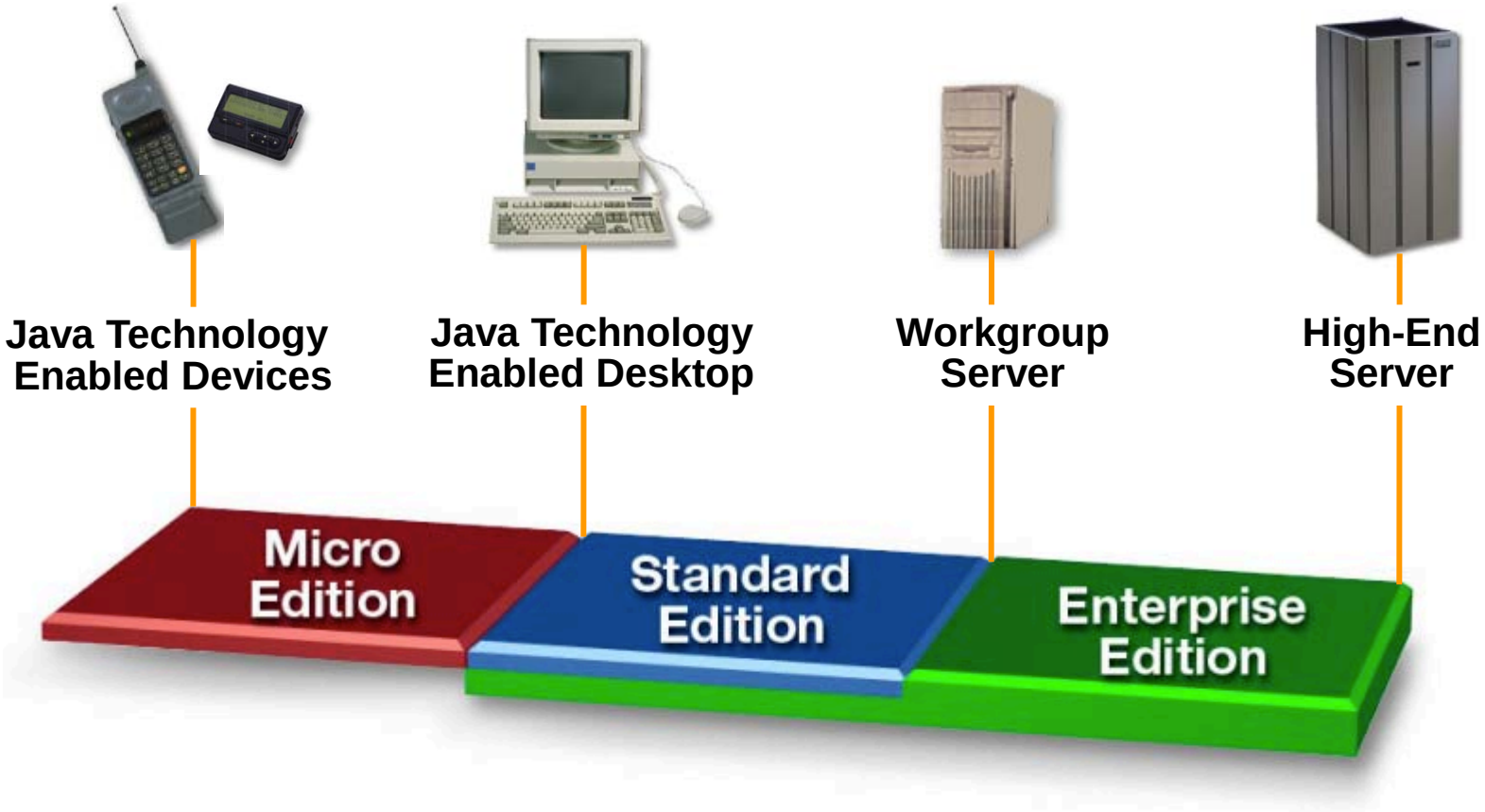
Προϊόντα

- App Servers
- Web Servers
- Components
- Databases
- Object to DB tools

Τι είναι το Java EE?

- Μια ανοικτή πρότυπη πλατφόρμα για
  - Ανάπτυξη, εκτέλεση και διαχείριση
  - n-tier, Web-enabled, server-centric, και component-based εφαρμογών

The Java™ Platform



The Java™ Platform

Optional Packages

Optional Packages

Java 2 Enterprise Edition (J2EE)

Java 2 Standard Edition (J2SE)

Personal Basis Profile

Personal Profile

Foundation Profile

MIDP

CDC

CLDC

Java Card APIs

JVM

KVM

CardVM

Java EE APIs

J2EE APIs και τεχνολογίες

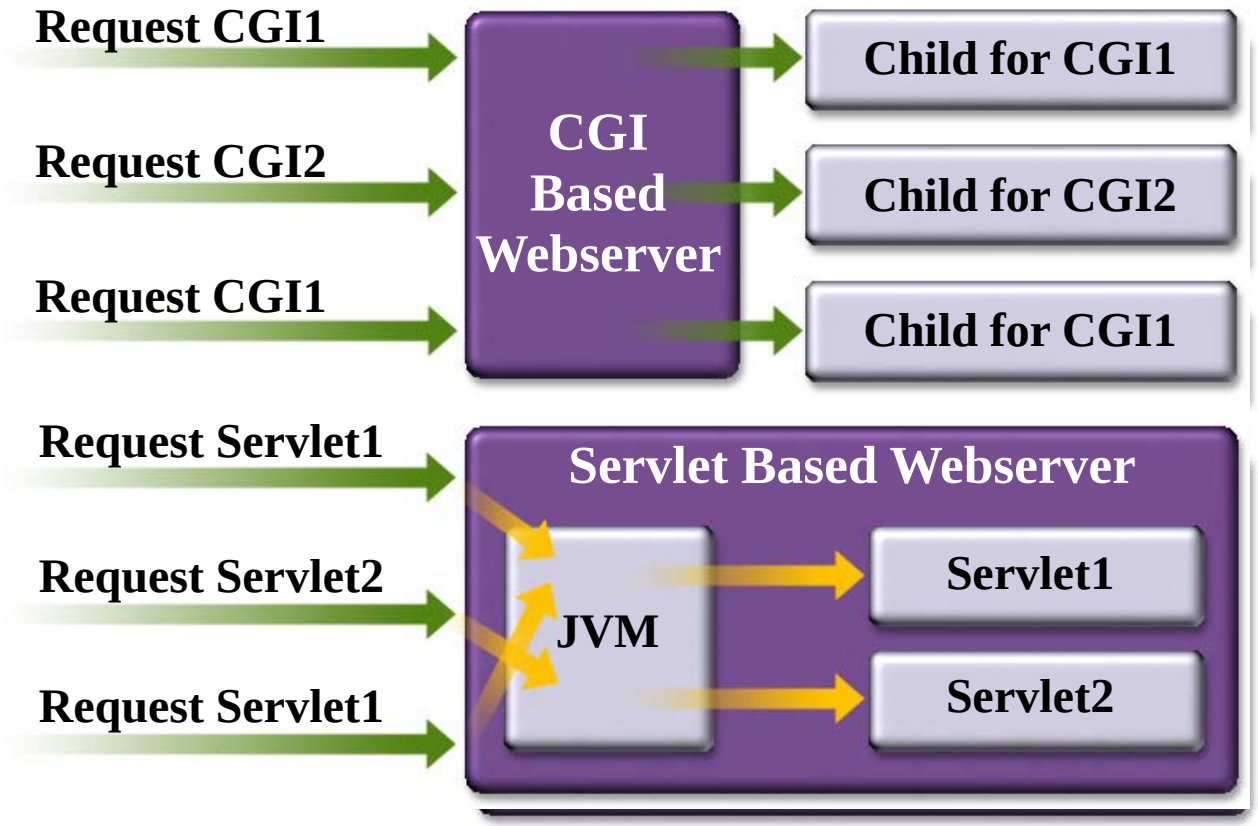
- Servlets
- JSP
- EJB 3.0
- JavaSever Faces
- Java Persistence (JPA), Hibernate
- Spring Framework

Java Servlets

Server Side Java - Servlets

- Απαιτούν την εγκατάσταση του Java Web Server (παρέχεται από τη Sun). Άρα μπορούν να μεταφερθούν σε οποιαδήποτε πλατφόρμα διαθέτει Java Virtual Machine
- Η εγκατάσταση του JWS ξεκινά κάποιο service που δέχεται κλήσεις σε συγκεκριμένη θύρα του server.
- Τα Servlets είναι classes που μπορεί να δημιουργούν HTML κώδικα, βασισμένα σε κάποιες παραμέτρους.
- Ο browser καλεί τα servlets με κάποιες παραμέτρους και παίρνει σαν απάντηση μια δημιουργημένη HTML σελίδα.

Servlet vs. CGI



- Καλύτερη διαχείριση μνήμης
- Ταχύτερη απόκριση

Στο CGI κάθε HTTP request δημιουργεί ένα νέο process.
Στο μοντέλο των servlet, ένα νήμα δημιουργείται σε κάθε κλήση (εντός του ίδιου JVM), το οποίο μπορεί να εξυπηρετήσει και άλλες κλήσεις.

Servlets

- Είναι java εφαρμογές που εκτελούνται από ένα web server, σε αντίθεση με τα applets που εκτελούνται στον client.
- Τα servlets πολλές φορές χρησιμοποιούνται για να φτιάξουν δυναμικές HTML σελίδες.

1. Δίνει τα στοιχεία του π.χ. Ηρακλής, Βαρλάμης
2. Πατάει submit
3. Καλείται η διεύθυνση http://www.home.com/reply.class?name='Ηρακλής',surname='Βαρλάμης'

4. Η σελίδα παράγεται δυναμικά και στέλνεται στον client

HelloWorld Servlet

```
public class HelloWorldServlet extends HttpServlet {
public void doGet (HttpServletRequest req,
HttpServletResponse res)
    throws ServletException, IOException  {
   PrintWriter           out;
   res.setContentType("text/html");
   out = res.getWriter();
   out.println("<html>");
   out.println("<head><title>Hello
World</title></head>");
   out.println("<body>");
   out.println("<h1>Hello World</h1>");
   out.println("</body></html>");  }
}
```

- Όταν στο browser δώσουμε διεύθυνση: www.server.com:4444/HelloWorld.class μας επιστρέφεται μια HTML σελίδα.

JSP

Java Server Pages

JSP – Java Server Pages

- Είναι HTML σελίδες που περιέχουν κώδικα Java. Ο κώδικας αυτός εκτελείται στο server πριν η σελίδα σταλεί στο χρήστη.
- Είναι πιο εύκολα στη χρήση από τα servlets.
- Και στις δύο περιπτώσεις ο web server πρέπει να υποστηρίζει Java δυνατότητες.
- π.χ. Πάνω από τον IIS, μπορούμε να βάλουμε τον Tomcat της Apache που είναι java web server

What is JSP Technology?

- Διαχωρίζουν το business logic από την παρουσίαση
  - Η παρουσίαση γίνεται με HTML (XML/XSLT)
  - Το business logic υλοποιείται με παραμετροποίηση έτοιμων Java Beans ή με custom tags
  - Καλύτερη διαχείριση κώδικα, επαναχρησιμοποίηση

example.jsp

```
<html> <head> <title> JSP Example</title>
<%@ page import = "java.io.*" %>
 </head>
<body>
<%
        String sn1; String sn2; int n1,n2;
        sn1 = request.getParameter("n1");
        sn2 = request.getParameter("n2");
        n1 = Integer.parseInt(sn1);
        n2 = Integer.parseInt(sn2);
        out.println("The numbers were " + sn1 + " and " + sn2);
        out.println("<br>");
        out.println("The sum is " + (n1+n2));
%>
</body>
</html>
```

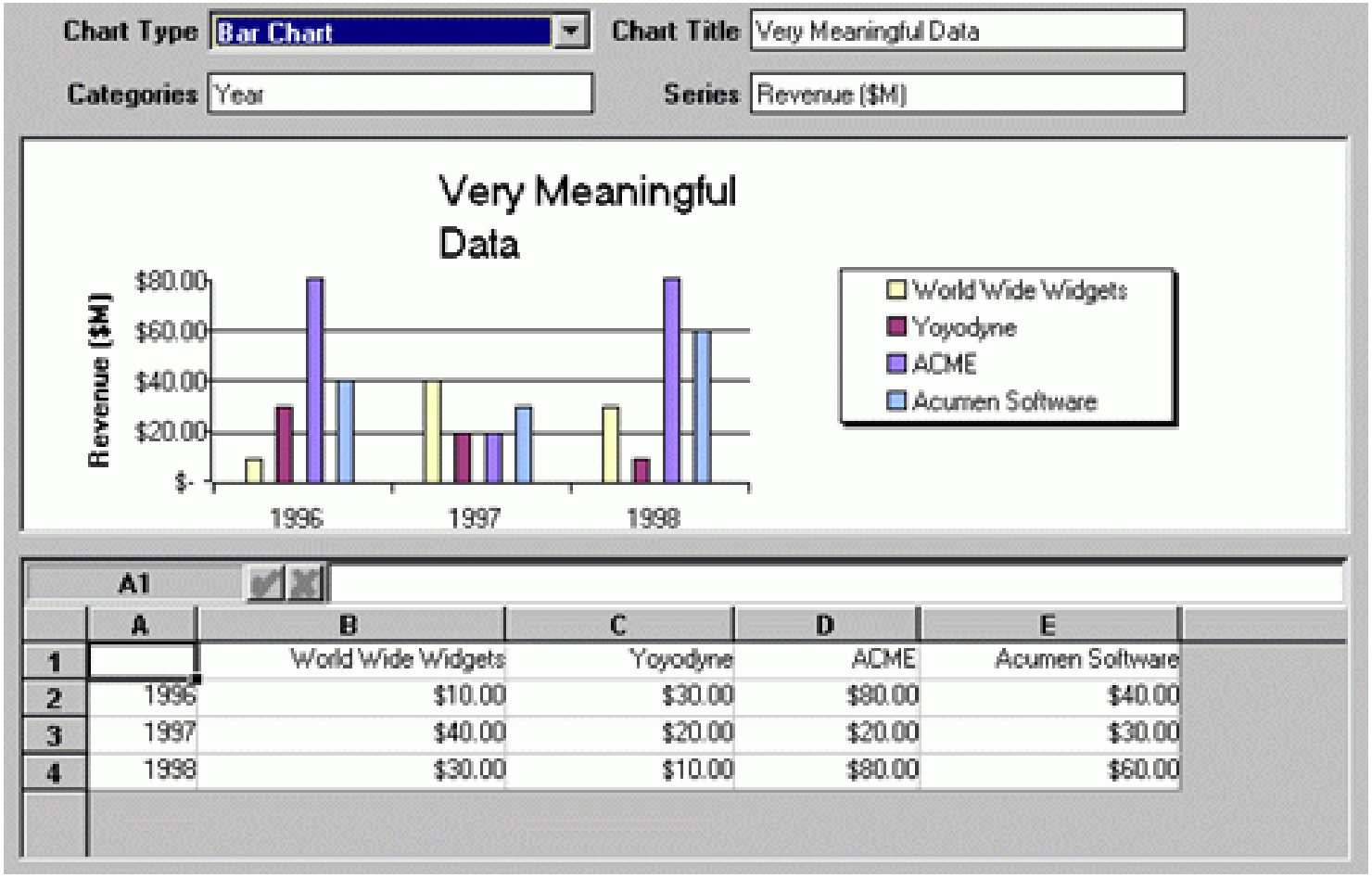
Το jsp θα κληθεί ως εξής http://www...../example.jsp?n1=1&n2=1

Java Beans

Java Beans

- Επαναχρησιμοποιήσιμα προγράμματα.
- Αποτελούν την απάντηση της Java στα ActiveX components.
- Είναι σύνθετα αντικείμενα (π.χ. κουμπιά, μπάρες κύλισης, ημερολόγια, επεξεργαστές κειμένου, λογιστικά φύλλα κλπ.) η συμπεριφορά των οποίων μπορεί να καθοριστεί.
- Τα Java Beans μπορούν να αγοραστούν και να χρησιμοποιηθούν μέσα από ένα γραφικό περιβάλλον ανάπτυξης εφαρμογών, χωρίς ειδικές γνώσεις προγραμματισμού.

Παράδειγμα (Barchart bean)



HTML+Java Beans

- Τα JSPs επιτρέπουν την ανάμειξη του HTML και του κώδικα της Java σ' έναν ενοποιημένο πηγαίο κώδικα
- Από τεχνική άποψη, μπορούμε να κάνουμε σ' ένα JSP όλη τη δουλειά ενός JavaBean
- Αλλά στην πραγματικότητα τα JSPs σχεδιάσθηκαν για να χρησιμοποιηθούν στο επίπεδο παρουσίασης
- Ο πολύ μεγάλος φόρτος υπολογισμών που συμβαίνει στο παρασκήνιο θα πρέπει να χειρισθεί από κάποιον μηχανισμό που να είναι αφοσιωμένος σ' αυτή τη δουλειά, όπως είναι τα JavaBeans

Παράδειγμα

```
package MyBeans;
 public class Circle {
            double radius;
            double area;
            public void setRadius(double r) {radius = r;}
            public double getRadius() {return radius;}
            public void setArea(double a) {area = a;}
            public double getArea() {return area;}
          }
```

```
<jsp:useBean name="circle1" class=“MyBeans.Circle">
<html>
<head>
<title> Ένα Απλό Παράδειγμα JavaBean </title>
</head>
<body>
        <jsp:setProperty name="circle1" property="radius" value="2">
        Εμβαδόν = <jsp:getProperty name="circle1" property="area">
</body>
</html>
```

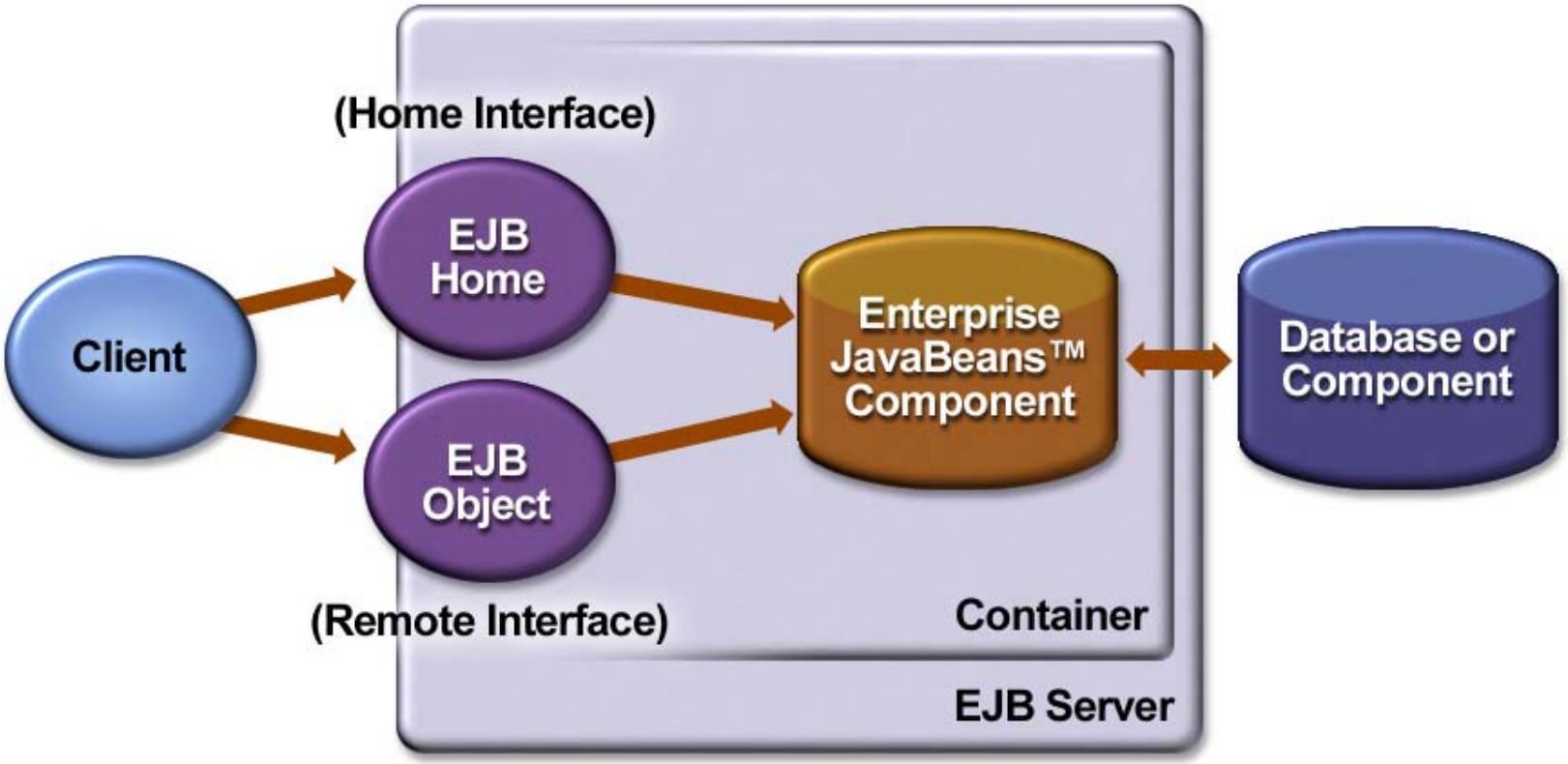
EJB

Enterprise Java Beans

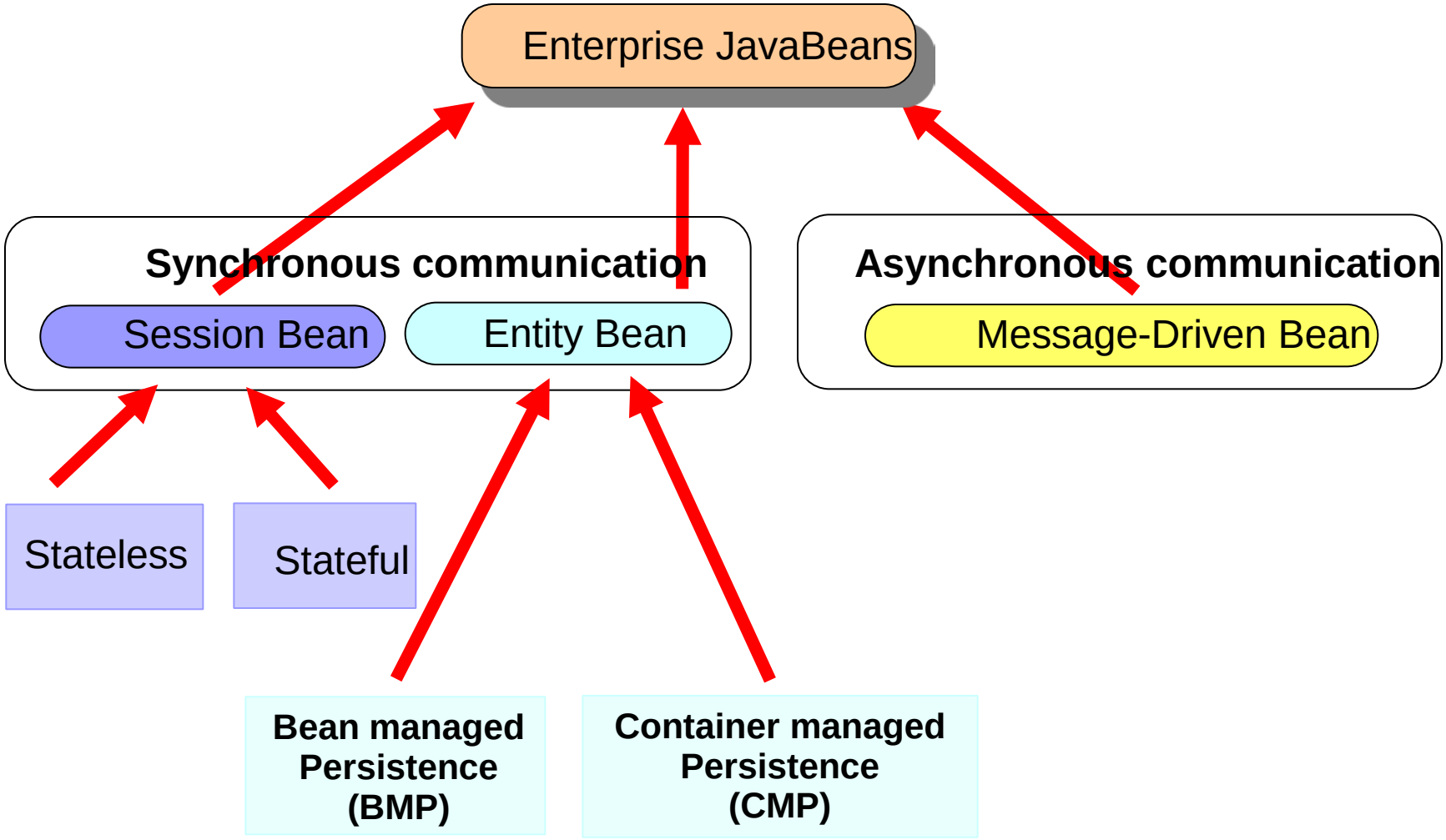
Τι είναι τα EJB

- Μια τεχνολογία που χρησιμοποιεί ολοκληρωμένες εφαρμογές (παραμετρικές) στο server (server-side component) για να λύσει συνηθισμένα προβλήματα
- Προσφέρουν εύκολη/γρήγορη ανάπτυξη εφαρμογών: Transactional, distributed, multi-tier, portable, scalable, secure, …
- Διαχωρίζει το business logic από το application logic (από το που και πως τρέχουν οι εφαρμογές)

EJB Architecture

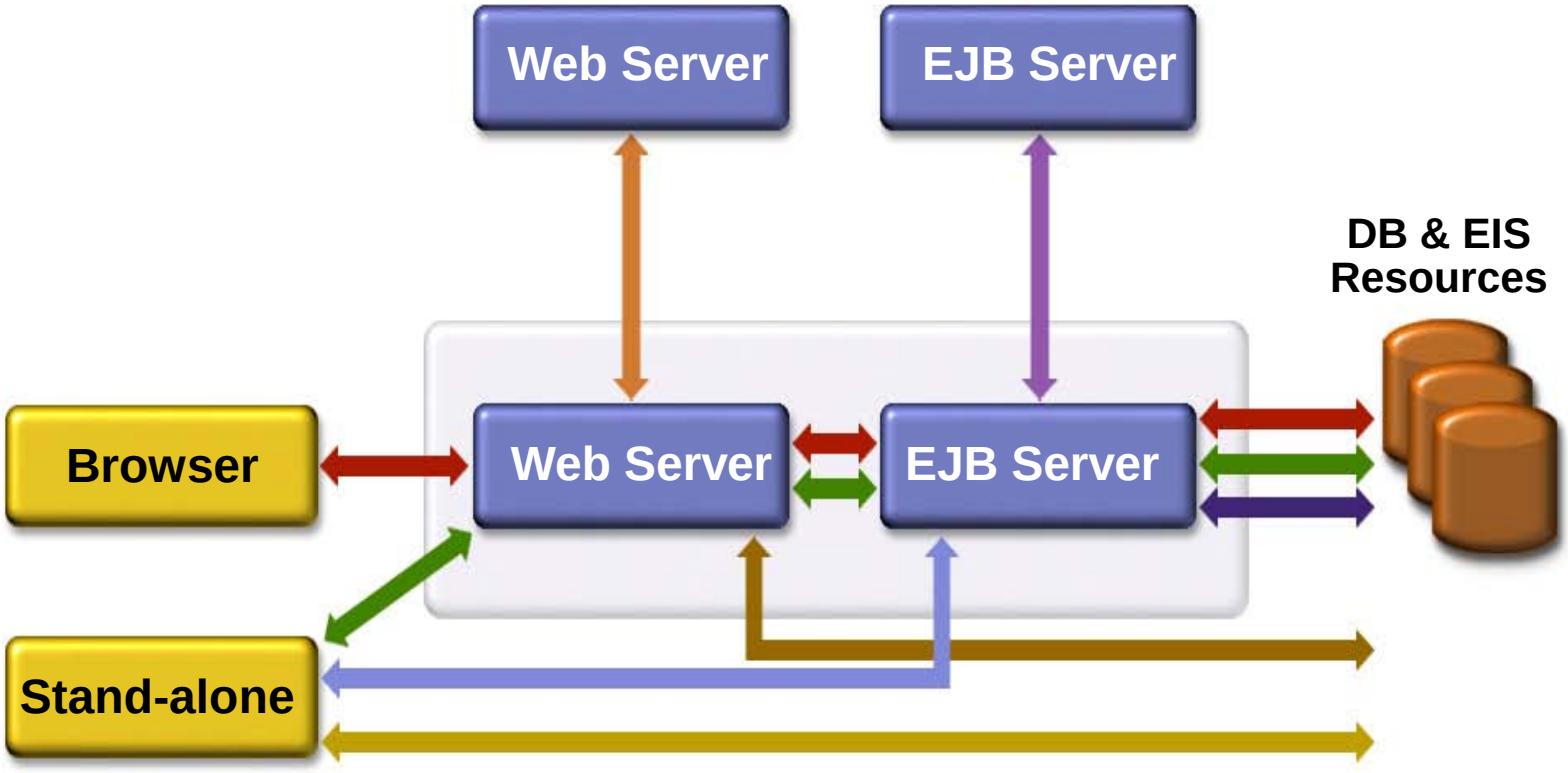


Enterprise JavaBeans

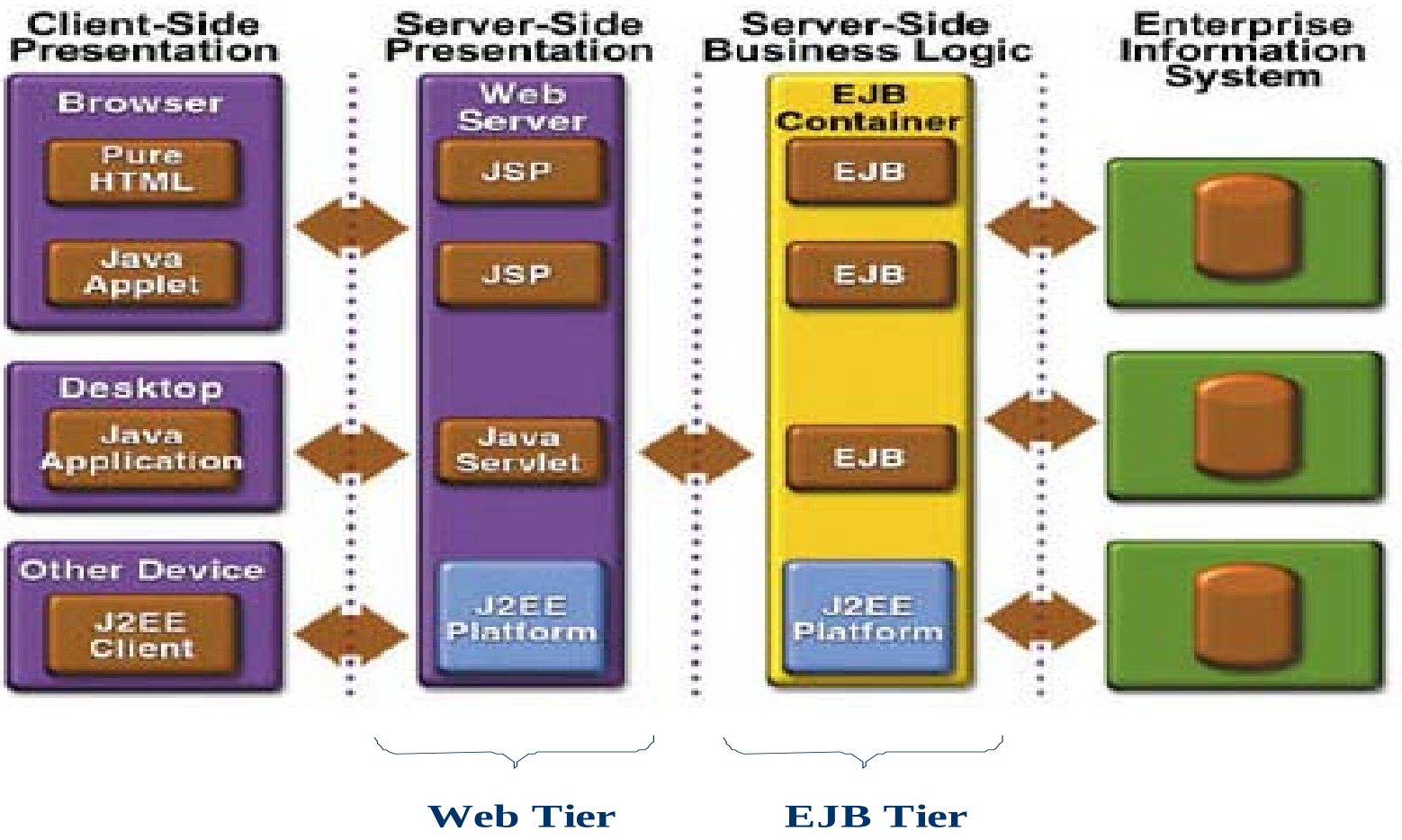


Συνολικά

Ενδεικτική αρχιτεκτονική μιας Java EE εφαρμογής



N-tier J2EE Architecture



J2EE Application Anatomies

- 4-tier J2EE applications
  - HTML client, JSP/Servlets, EJB, JDBC/Connector
- 3-tier J2EE applications
  - HTML client, JSP/Servlets, JDBC
- 3-tier J2EE applications
  - EJB standalone applications, EJB, JDBC/Connector
- B2B Enterprise applications
  - J2EE platform to J2EE platform through the exchange of JMS or XML-based messages