Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Προηγμένες Μέθοδοι Προγραμματισμού

ΠΜΣ 2022-23 (M135.CS1E, M135.CS23B, M135.IC1E, παλαιό: M117)

Πρότυπα σχεδίασης (4)

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Περιεχόμενα

- Δομικά πρότυπα σχεδίασης
- Πρότυπα σχεδίασης συμπεριφοράς

Δομικά πρότυπα σχεδίασης

Πρότυπα σχετικά με τη δομή και τη συσχέτιση κλάσεων

Περιεχόμενα

Adapter (Wrapper)

Σκοπός: Να κάνει μια κλάση συμβατή με κάποιο interface που αναμένει ένας χρήστης της

Μια υπάρχουσα κλάση γίνεται συμβατή με ένα νέο interface

Παράδειγμα

java.lang.Runnable

```
public interface Runnable {
  public void run()
}
```

java.util.concurrent.Callable

```
public interface Callable<V> {
  public V call() throws Exception
}
```

```java
public class RunnableAsCallable implements Callable<Void> {
  private final Runnable runnable;

  public RunnableAsCallable(Runnable runnable) {
    this.runnable = runnable;
  }

  @Override
  public Void call() throws Exception{
    runnable.run();
    return null;
  }
}
```

Συζήτηση

- Μηχανισμός για να κάνουμε δύο ξεχωριστά συστατικά (με διαφορετικό σχεδιασμό) να λειτουργήσουν μαζί, αφού έχουν υλοποιηθεί.
- "Ντύνουμε" ένα interface με τα ρούχα ένός άλλου.

Bridge

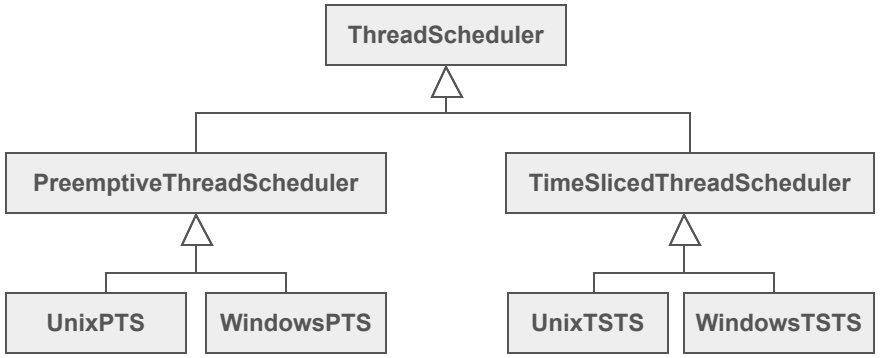
Σκοπός: Να διαχωρίσουμε την αφαίρεση (interface) από την υλοποίηση (class), ώστε να χρησιμοποιούνται ανεξάρτητα.

Παράδειγμα

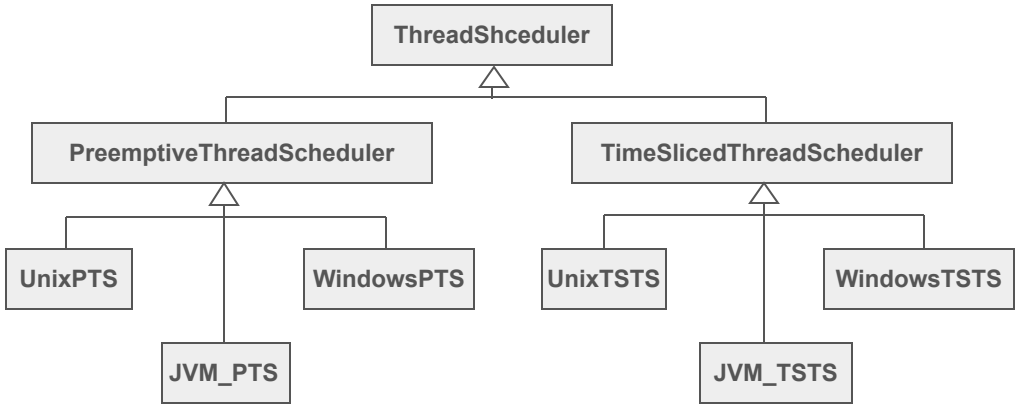
Δύο βασικές έννοιες

- Thread scheduler
- Platform

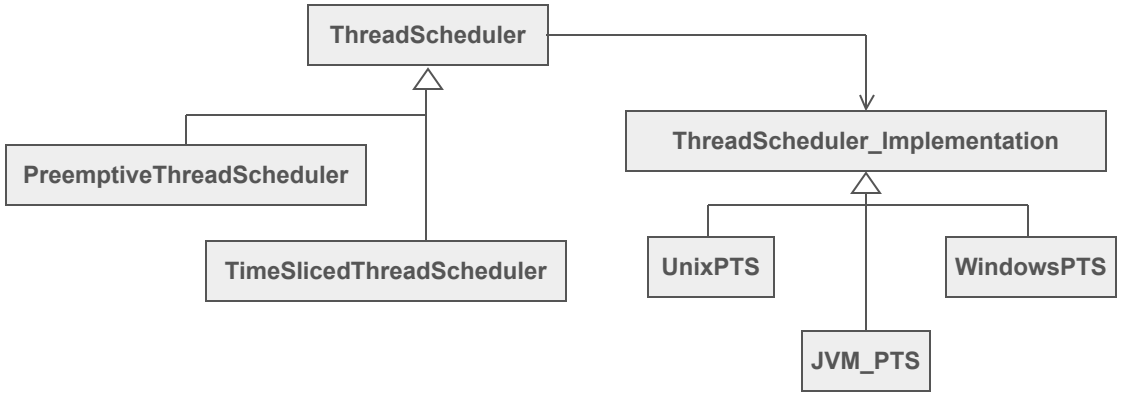
Παράδειγμα (Subclassing)



Παράδειγμα (Subclassing)



Παράδειγμα (Bridge)



Συζήτηση

- Έμφαση στη σύνθεση και όχι στην κληρονομικότητα
- Χρήση δύο ιεραρχιών κληρονομικότητας: μία "δημόσια" και μία "ιδιωτική"
- Οι "δημόσιες" αφαιρέσεις χρησιμοποιούν τις "ιδιωτικές" υλοποιήσεις
- Οι δύο ιεραρχίες μπορούν να εξελιχθούν ανεξάρτητα μεταξύ τους

Composite

Σκοπός: Να επιτρέψουμε σε μια κλάση-χρήστη να χειρίζεται απλά και σύνθετα αντικείμενα με ομοιόμορφο τρόπο

Παράδειγμα

```
interface Resolver {
  File resolve(String text) throws NotFoundException;
}

class LocalFileResolver implements Resolver {
  ...
}

class URLResolver implements Resolver {
  ...
}
```

```
class CompositeResolver implements Resolver {
  private final Resolver[] resolvers;
  CompositeResolver(Resolver... r) {
    this.resolvers = r;
  }
  File resolve(text) throws NotFoundException {
   File f = null;
   for(resolver: resolvers) {
     try {
       f = resolver.resolve(text);
     }
     catch(NotFoundException nfe) {
      ..
     }
     if (f == null) {
       throw new NotFoundException(text);
     }
     else {
       return f;
     }
   }
  }
}
```

Decorator

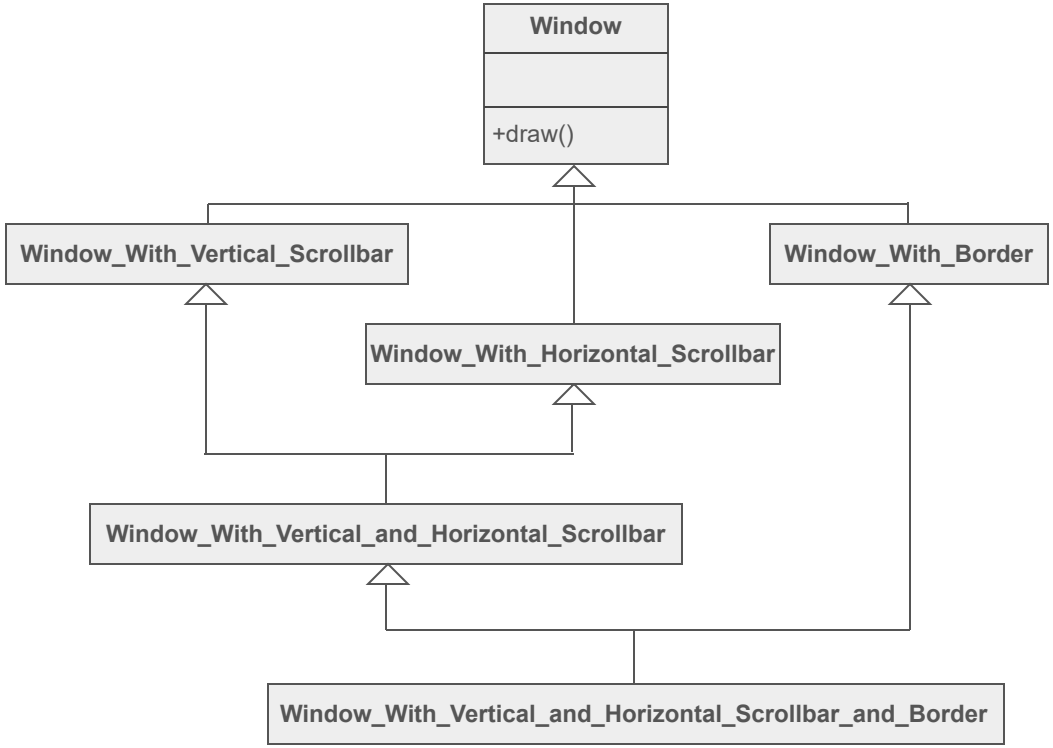
Σκοπός: Να προσθέσουμε επιλεκτικές και προαιρετικές δυνατότητες σε ένα αντικείμενο χωρίς κληρονομικότητα

Παράδειγμα

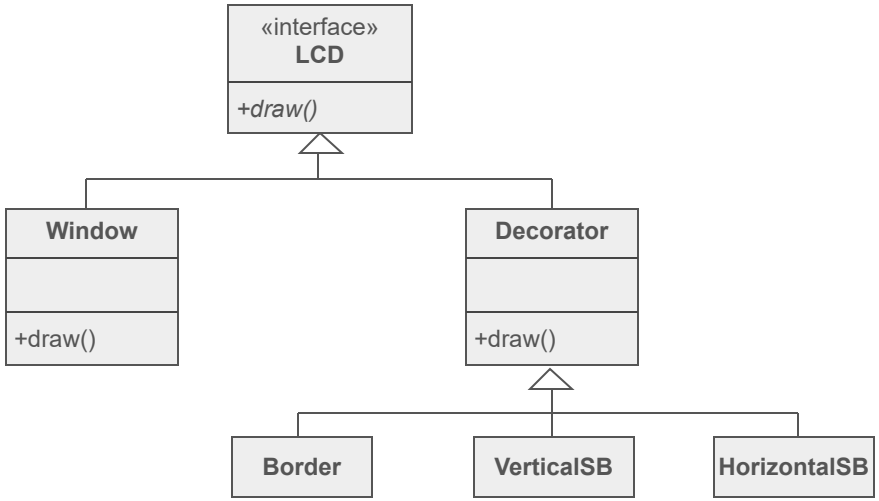
Τέσσερις βασικές έννοιες

- Window
- Horizontal Scrollbar
- Vertical Scrollbar
- Border

Παράδειγμα (Subclassing)



Παράδειγμα (Decorator)



```
Widget w = new BorderDecorator(
  new HorizontalScrollBarDecorator(
    new VerticalScrollBarDecorator(
      new Window("Title")
    )
  )
);
w.draw();
```

Συζήτηση

- Παρέχει μια πιο ευέλικτη εναλλακτική για την προσθήκη λειτουργικότητας σε μια κλάση σε σχέση με το subclassing (κληρονομικότητα)
- Recursive wrapping

Πραγματικό παράδειγμα (Java IO)

```
OutputStream os = new BufferedOutputStream(
  new FileOutputStream("/path/to/file")
);
```

Facade

Σκοπός: Να παρασχεθεί ένα απλό interface σε ένα σύνθετο υποσύστημα.

Παράδειγμα

```
class VideoManagerFacade {

  File convert(String fileName, String format) {
    VideoFile file = new VideoFile(fileName);
    Codec srcCodec = CodecFactory.of(file);
    Codec destCodec;
    if (format.equals("ogg")) {
      destCodec = new OggCompressionCodec();
    } else {
      destCodec = new MPEG4CompressionCodec();
    }
    VideoFile buffer = BitrateReader.read(file, srcCodec);
    VideoFile tmp = BitrateReader.convert(buffer, destCodec);
    File result = (new AudioMixer()).fix(tmp);
    return result;
  }

}
```

Proxy

Σκοπός: Να προσφέρει έναν ενδιάμεσο για ένα άλλο αντικείμενο, ελέγχοντας την πρόσβαση σε αυτό

Εφαρμογές

- Οκνηρή αρχικοποίηση (virtual proxy)
- Έλεγχος πρόσβασης (access proxy)
- Εκτέλεση απομακρυσμένων υπηρεσιών (remote proxy)
- Χρήση ενδιάμεσης μνήμη (caching)
- Καταγραφή ενεργειών (logging proxy)

Παράδειγμα

```
class Router {
  void setup() {
    route("/index.html", new PublicResource())
    route("/cart.html", new ProtectedResource())
  }

  void service(HttpRequest req, HttpResponse res) {
     MVCResource res = getResourceFor(req)
     LoggingProxy proxy = new LoggingProxy(res)
     proxy.doService(req, res)
  }
}
```

```
class LoggingProxy extends MVCResource {
  private static final Logger log = ...
  private final MVCResource resource;
  LoggingProxy(MVCResource resource) {
    this.resource = resource;
  }
  User createUser(HttpRequest req) {
    return res.createUser
  }
  void doService(HttpRequest req, HttpResponse res) {
    log.log("Starting service of $resource")
    long t = System.currentTimeMillis()
    try {
      resource.doService(req, res)
    }
    finally {
      t = System.currentTimeMillis() - t
      log.log("Ending service of $resource (duration $t ms)")
    }
  }
}
```

Πρότυπα σχεδίασης συμπεριφοράς

Πρότυπα σχετικά με τη συμπεριφορά / επικοινωνία των αντικειμένων

Περιεχόμενα

Επίσης

8. Iterator / Iterable
9. Observer / Observable
10. Callback / Future / Promise

Που τα είδαμε σε παλιότερη διάλεξη

Visitor

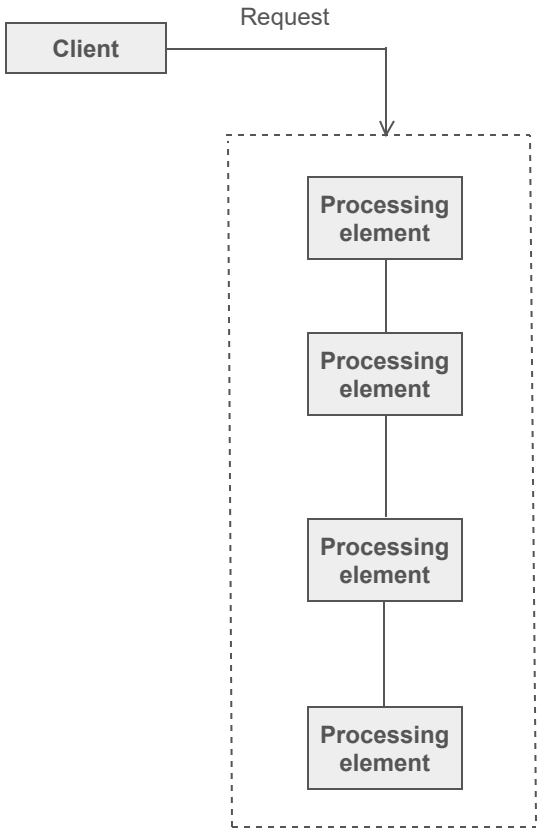
Σκοπός: Να διαχωρίσει τον αλγόριθμο από τη δομή δεδομένων.

Η "λογική" της επίσκεψης στα στοιχεία της δομής (traversal) διαχωρίζεται από τη δομή αυτή καθ' αυτή.

Το έχουμε δει ξανά στην εισαγωγή για τα πρότυπα σχεδίασης

Chain of responsibility

Σκοπός: Να περάσει ένα αίτημα μέσω μιας αλυσίδας αντικειμένων


Client
Request
Processing element
Processing element
Processing element
Processing element

Παράδειγμα

Το παράδειγμα του Composite προτύπου από το προηγούμενο μάθημα χρησιμοποιεί το Chain of responsibility

Πραγματικό παράδειγμα

- Servlet filters
- Express.js/Node.js middleware

Command

Σκοπός: Να "ενθυλακώσει" ένα αίτημα ως αντικείμενο, διαχωρίζοντας τον αποστολέα από τον παραλήπτη

Παράδειγμα

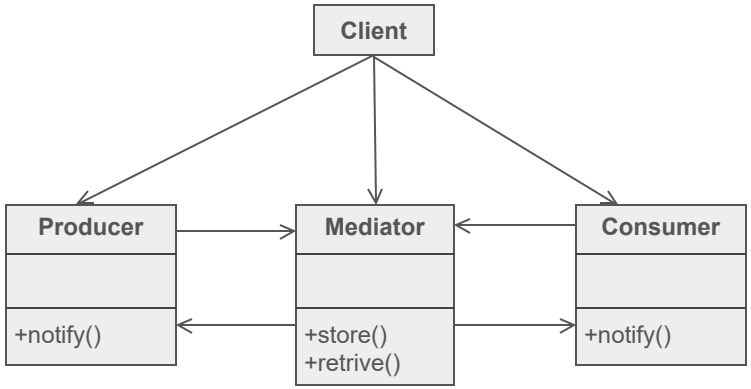
java.lang.Runnable

java.util.concurrent.Callable

Mediator

Σκοπός: Να δημιουργήσουμε έναν "ενδιάμεσο" που "γνωρίζει" πώς αλληλεπιδρούν διάφορα αντικείμενα

Χρήσιμο για σχέσεις N-N


Client
Producer
+notify()
Mediator
+store()
+retrive()
Consumer
+notify()

Memento

Σκοπός: Να "εξωτερικεύσουμε" με ασφαλή τρόπο την εσωτερική κατάσταση ενός αντικειμένου ώστε να μπορούμε να την επαναφέρουμε αργότερα.

Για την υλοποίηση λειτουργιών undo ή rollback

Παράδειγμα

```
class Memento {
  private final String state;

  public Memento(String state) {
      this.state = state;
  }

  public String getState() {
      return state;
  }
}
```

```
class Originator {
  private String state;
  ...//more fields that "depend" on the state

  private void setState(String state) {
    this.state = state;
    //change fields depending on the state
  }

  public Memento save() {
    return new Memento(state);
  }

  public void restore(Memento m) {
    setState(m.getState());
  }
}
```

Strategy

Σκοπός: Να αναπαραστήσουμε μια οικογένεια αλγορίθμων με ομοιόμορφο τρόπο ώστε να τους εναλλάσσουμε

Παράδειγμα

```
class Client {
  String id
  //...
  BillingStrategy billingPlan
}

interface BillingStrategy {
  //Fundamental billing functions
}

class EnterprisePlan implements BillingStrategy {
  //Billing details of the enterprise plan
}

class SimplePlan implements BillingStrategy {
  //Billing details of the simple plan
}
```

Null object

Σκοπός: Να υλοποιήσουμε με ομοιόμορφο τρόπο το NO-OP (do nothing)

Αποφεύγουμε τους ελέγχους για null διατηρώντας τη λογική του αλγορίθμου απλή και καθαρή

Παράδειγμα

Ένα ενδεχόμενο FreePlan στην περίπτωση του BillingStrategy παραδείγματος μπορεί να υλοποιηθεί ως Null object (μια υλοποίηση του BillingStrategy με μεθόδους που δεν κάνουν τίποτα)!