ΗΥ352 : ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

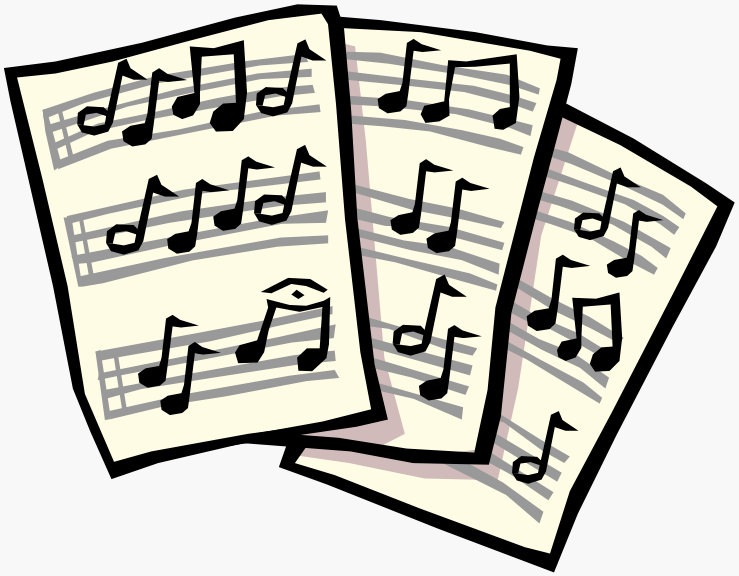
ΔΙΔΑΣΚΩΝ

Αντώνιος Σαββίδης

ΕΝΟΤΗΤΑ 5

ΣΧΕΔΙΑΣΤΙΚΑ ΠΡΟΤΥΠΑ

Αριθμός διαλέξεων 5 – Διάλεξη 4η



Περιεχόμενα

- *Progress monitoring*
- Listener
- View
- Decorator

Progress monitoring (1/6)

- ***Πρόβλημα***
  - Υπάρχουν χρονοβόρες συναρτήσεις. Χρειάζεται να λαμβάνει τακτικά το πρόγραμμά μας ειδοποιήσεις για την εξέλιξη της διαδικασίας, για να, π.χ., παρουσιάσει μία «μπάρα προόδου».
  - Πρέπει να μπορούμε να διακόψουμε, ή να παγώσουμε και να επαναφέρουμε τέτοιες λειτουργίες, ανά πάσα στιγμή.

Progress monitoring (2/6)

- ***Λύση***
  - Κάθε τέτοια συνάρτηση επιστρέφει ένα στιγμιότυπο ενός ειδικού API ελέγχου και παρακολούθησης προόδου, αντί να εκτελεί απλώς την συγκεκριμένη διαδικασία εσωτερικά.
  - Ο χρήστης του API, μπορεί να: αρχίσει, τερματίσει, παγώσει, συνεχίσει, την διαδικασία. Όταν τελειώσει η διαδικασία, το στιγμιότυπο καταστρέφεται από τον χρήστη.

Progress monitoring (3/6)

- ***Επιπτώσεις***
  - Κάθε τέτοια λειτουργία μπορεί να ελέγχεται ανεξάρτητα, ενώ μπορεί να έχουμε πολλές τέτοιες κλήσεις «παράλληλα». Εάν οι λειτουργίες πρέπει να μπορούν να καλούνται από ανεξάρτητα threads, πρέπει να υλοποιηθούν με thread safe τρόπο.
  - Οι κληρονόμοι κλάσεις του progress monitoring API (μία ανά λειτουργία) πρέπει να περιέχουν μεταβλητές για την κατάσταση εκτέλεσης της λειτουργίας.

Progress monitoring (4/6)

Παράδειγμα (1/3)

```
typedef void (*ProgressMonitorFunc)(
    ProgressMonitoring* progressInst,
    void* closure
);

class ProgressMonitoring {
public:
 virtual unsigned  GetTotalUnits (void) const = 0;
 virtual unsigned  GetUnitsDone (void) const = 0;
 virtual void      Stop (void) = 0;
 virtual void      Suspend (void) = 0;
 virtual void      Resume (void) = 0;
 virtual void      AddMonitor (ProgressMonitorFunc f, void* closure);
 virtual void      RemoveMonitor (ProgressMonitorFunc f, void* closure);
 virtual void      Work (unsigned unitsToDo) = 0;
};
```

Progress monitoring (5/6)

Παράδειγμα (2/3)

```
#define ALL_UNITS_AT_ONCE   0
class InstructionLoader: public ProgressMonitoring {
 private:
   FILE*           fp;
   unsigned int    total;
   unsigned int    curr;
   bool            isSuspended;
   VirtualMachine* vm;
   bool            ReadNext (void);
 public:
   virtual unsigned int GetUnitsDone (void) const override { return curr; }
   virtual unsigned int GetTotalUnits (void) const override { return total; }
   virtual void         Work (unsigned int totalToRead) override;
   InstructionLoader (const char* filePath, VirtualMachine* _vm) {
      FILE* fp = fopen(filePath, "r");
      fread(&total, sizeof(unsigned int), 1, fp); // assume total is first
      curr = 0; isSuspended = false; vm = _vm;
   }
};
```

Μεταβλητές καταστάσεως και προόδου λειτουργίας

Στιγμιότυπο κλάσης που υποστηρίζει τη λειτουργία

Progress monitoring (6/6)

Παράδειγμα (3/3)

```
class VirtualMachine {
   friend class InstructionLoader; // πιθανόν να χρειαστεί το friendship
   public:
   InstructionLoader* Load (const char* file) {
      return new InstructionLoader(file, this);
   }
};
…
static void DisplayProgressBar (ProgressMonitoring* p, void* unused) {
   unsigned percentDone = (p->GetUnitsDone() * 100) / p->GetTotalUnits();
   <Ζωγράφισε τη μπάρα προόδου για το percentDone>;
}
...
Virtualmachine vm;
InstructionLoader* loader = vm.Load("<some path here>");
loader->AddMonitor(DisplayProgressBar, (void*) 0);
loader->Work(ALL_UNITS_AT_ONCE); // Η Work εσωτερικά χρησιμοποιεί το vm
delete loader;
```

Περιεχόμενα

- Progress monitoring
- *Listener*
- View
- Decorator

Listener (1/7)

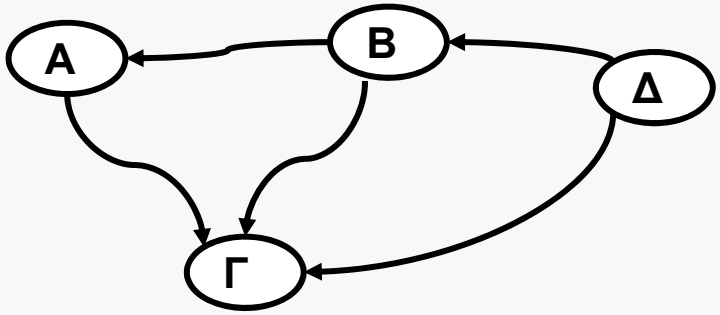
- ***Πρόβλημα***
  - Υπάρχουν χαρακτηριστικά στιγμιότυπων κάποιας κλάσης *A* τα οποία μπορεί να μεταβάλλονται σε διάφορα σημεία από το client πρόγραμμα, ενώ υπάρχουν στιγμιότυπα άλλης κλάσης *B* τα οποία θα πρέπει να πληροφορηθούν άμεσα για τέτοιου είδους αλλαγές

Listener (2/7)

- ***Λύση***
  - Η κλάση A παρέχει ένα API για την «εκδήλωση ενδιαφέροντος» από τους clients, ώστε αυτοί να ειδοποιούνται κατάλληλα σε περίπτωση τροποποίησης.
    - Στην πράξη κατά την εκδήλωση ενδιαφέροντος ο client πρέπει να περάσει ως παράμετρο ή δείκτη σε συνάρτηση, ή ένα στιγμιότυπο κατάλληλης *functor* κλάσης, δηλ. κλάσης με υπερφορτωμένο τον τελεστή ().
  - Λέγεται και *Observer*

Listener (3/7)

- ***Επιπτώσεις***
  - Ο client δεν χρειάζεται να ελέγχει τακτικά πότε το εκάστοτε ενδιαφέρον στιγμιότυπο μεταβάλλεται
  - Πολλοί clients μπορεί να ειδοποιούνται ασύγχρονα για την μεταβολή διαφόρων χαρακτηριστικών του στιγμιότυπου
  - Πολύπλοκα σχήματα εξάρτησης στιγμιότυπων μπορούν να εκφραστούν και να υλοποιηθούν με αυτό τον τρόπο, χωρίς αυτό να συνεπάγεται και εξάρτηση υλοποίησης των κλάσεων



- **A listens Γ**
- **B listens A**
- **B listens Γ**
- **Δ listens B**
- **Δ listens Γ**

Listener (4/7)

Παράδειγμα (1/4)

```
class HumanoidClass {
};
```

Αυτή είναι η κλάση που περιέχει μεταβλητές για τα φυσικά χαρακτηριστικά: *ύψος, χρώμα μαλλιών, μήκος μαλλιών, σχήμα προσώπου, χρώμα ματιών, βάρος, σχήμα μύτης, σχήμα αυτιών, αναλογίες άκρων, ειδικά χαρακτηριστικά*, κλπ.

```
class HumanoidClassRenderer {
};
```

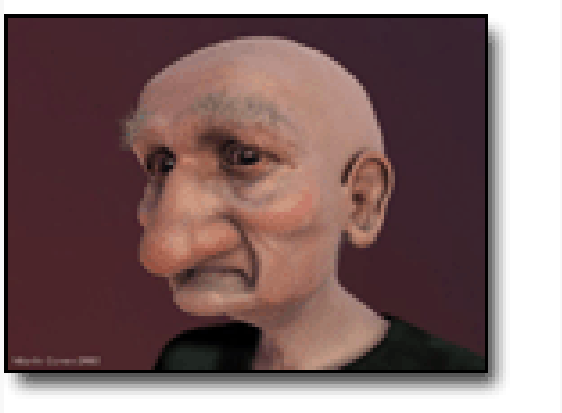
Αυτή η κλάση δέχεται ένα στιγμιότυπο HumanoidClass και ζωγραφίζει κατάλληλα ένα μοντέλο σύμφωνα με τις μεταβλητές φυσικών χαρακτηριστικών του στιγμιότυπου.

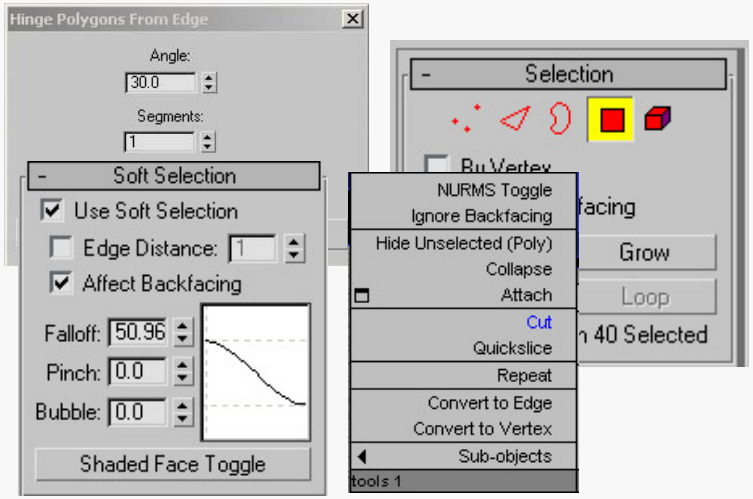
```
class HumanoidClassEditor {
};
```

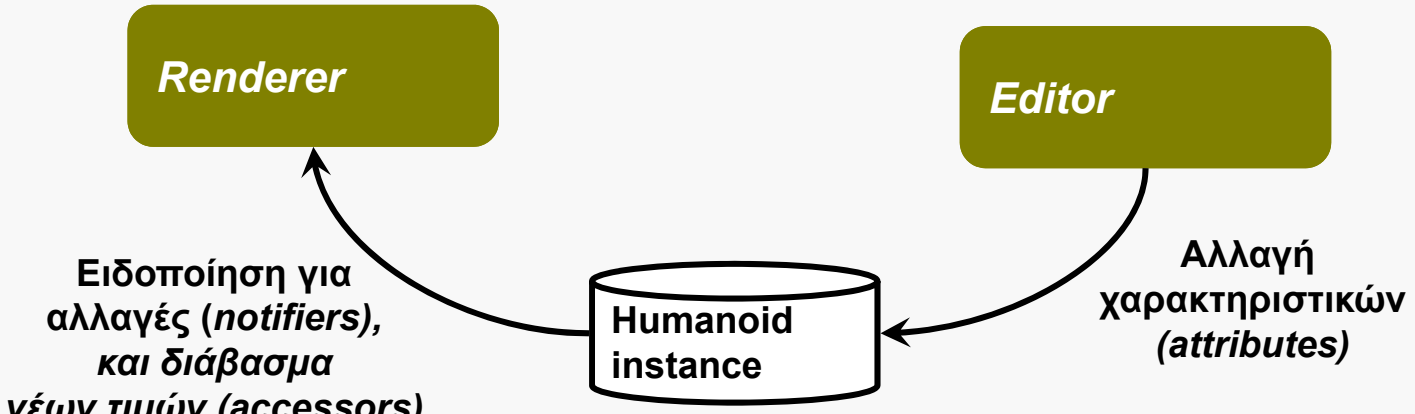
Αυτή η κλάση παρουσιάζει ένα απλό user interface για την αλλαγή τιμών στα φυσικά χαρακτηριστικά του στιγμιότυπου που δίνεται ως παράμετρος, με το οποίο ο χρήστης μπορεί να τα μεταβάλλει ανά πάσα στιγμή.

Listener (5/7)

Παράδειγμα (2/4)







Images are a courtesy of Discreet, and 3ds Max product

Listener (6/7)

Παράδειγμα (3/4)

```
class HumanoidListener {
   public:
   virtual void operator()(HumanoidClass* humanoid) const = 0;
   virtual HumanoidListener* Clone (void) const = 0;
};

class HumanoidClass {
  private:
    HairColor                                   hairColor;
    std::list<HumanoidListener*>                hairColorListeners1;
    std::map<unsigned, HumanoidListener*>   hairColorListeners2;
  public:
    // style-1: caller allocates and deletes
    void AddHairColorListener(HumanoidListener* f)
       { hairColorListeners1.push_back(f); }
    void RemoveHairColorListener(HumanoidListener* f)
       { hairColorListeners1.remove(f); }
    // style-2: no allocation needed, just use of unique tags
    void AddHairColorListener(const HumanoidListener& f, unsigned tag)
       { hairColorListeners2[tag] = f.Clone(); }
    void RemoveHairColorListener(unsigned tag)
       { hairColorListeners2.remove(tag); }

    void SetHairColor(HairColor color) {
       hairColor = color;
       CallHairColorListeners();
    }
};
```

Χρειάζεται μόνο εάν πρέπει να μπορεί ο caller αφαιρεί έναν listener αλλιώς είναι περιττό

Listener (7/7)

Παράδειγμα (4/4)

```
class HumanoidClassRenderer {
 private:
    class HairColorListener : public HumanoidListener {
    private:
        HumanoidClassRenderer* r;
    public:
        virtual void operator()(HumanoidClass* h)
            { r->Render(h); }
        HairColorListener (HumanoidClassRenderer* _r) : r(_r){}
      };
    HairColorListener* hairColorListener; // style-1
 public:
    void Render (HumanoidClass* humanoid);
        HumanoidClassRenderer (HumanoidClass* h) {
        h->AddHairColorListener(    // style-1
            hairColorListener = new HairColorListener(this)
        );
        h->AddHairColorListener(    // style-2
            HairColorListener(this), (unsigned) this
        );
    }
    ~HumanoidClassRenderer(){
        h->RemoveHairColorListener(hairColorListener); // style-1
        delete hairColorListener; // style-1
        h->RemoveHairColorListener((unsigned) this);    // style-2
    }
};
```

Ένθετο

```
// when you do not remove a listener just do this
void HumanoidClass::AddHairColorListener (
    const HumanoidListener& listener
) {
        haircolorListeners.push_back(
                listener.Clone()
        );
}

HumanoidClass::~HumanoidClass() {
        for (auto i : haircolorListeners)
                delete i;
}
```

Περιεχόμενα

- Progress monitoring
- Listener
- *View*
- Decorator

View (1/1)

- Το view pattern είναι ειδική περίπτωση listener που ο σκοπός του είναι να παρουσιάζει μία εικόνα του στιγμιότυπου σε άλλους clients (εάν ο client είναι το display τότε έχουμε ένα γραφικό view)
- Το προηγούμενο παράδειγμα είναι και μία περίπτωση του view pattern

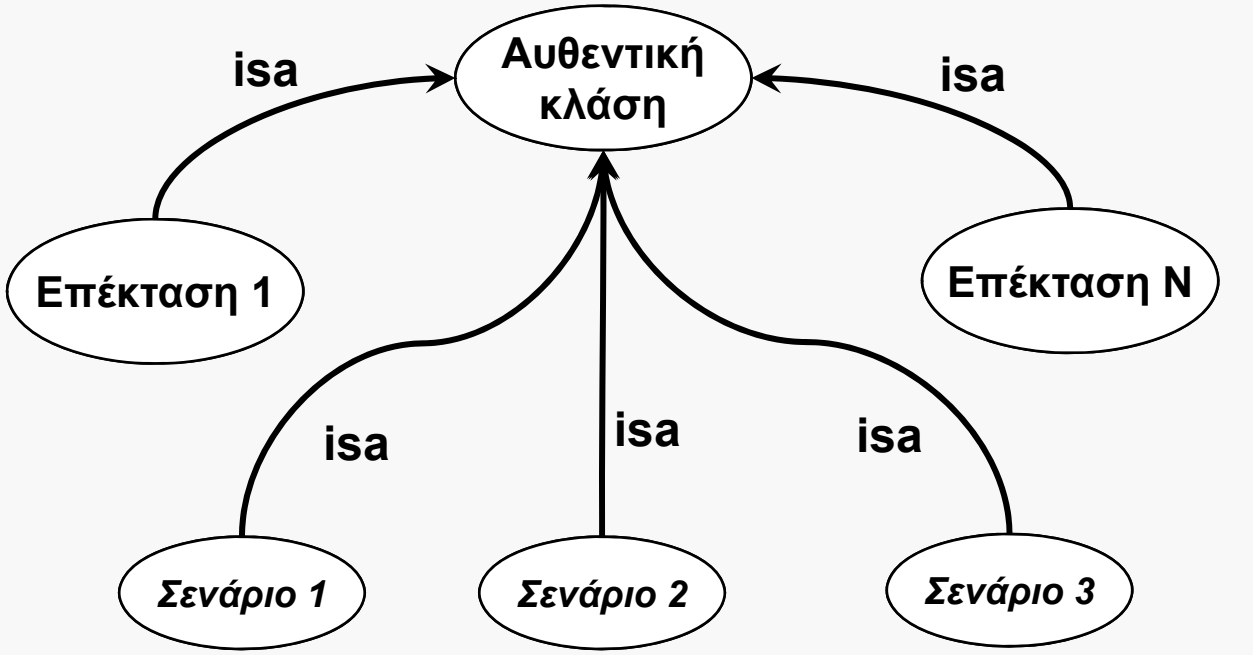
Περιεχόμενα

- Progress monitoring
- Listener
- View
- *Decorator*

Decorator (1/7)

- ***Πρόβλημα***
  - Έχουμε μία κλάση που μπορεί να επεκταθεί με πολλούς τρόπους ώστε να παρέχει επιπλέον λειτουργίες και χαρακτηριστικά, χωρίς να επηρεάζεται η αυθεντική λειτουργικότητα. Θέλουμε οι διάφορες επεκτάσεις να μπορούν να συνδυάζονται ευέλικτα.
  - Η χρήση κληρονομικότητας για κάθε δυνατή επέκταση δεν είναι πρόσφορη προσέγγιση, γιατί για την υποστήριξη συνδυασμών των επεκτάσεων θα πρέπει να οριστούν τόσες κλάσεις, όσοι και οι πιθανοί συνδυασμοί. Αντίθετα, ο «συνδυασμός» (mixing) θα ήταν καλύτερο να υποστηριχθεί και αυτός ως μία λειτουργία.

Decorator (2/7)

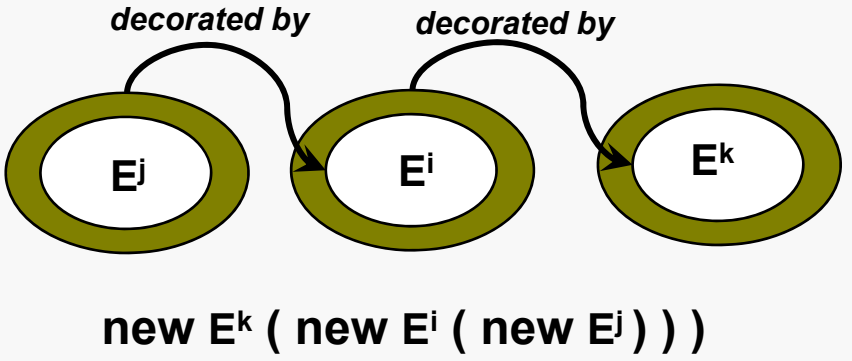


Decorator (3/7)

- ***Λύση***
  - Υλοποιούμε κανονικά κάθε επέκταση ως κληρονόμο της αυθεντικής κλάσης, υποστηρίζοντας πλήρως το αυθεντικό super-API, προσθέτοντας και ότι επιπλέον χαρακτηριστικά χρειάζονται για την επέκταση.
  - Η κλάση επέκτασης δέχεται ένα στιγμιότυπο – δείκτη της αυθεντικής κλάσης.
  - Η υλοποίηση των τροποποιημένων συναρτήσεων, αντί να εφαρμόζει την κλασική κλήση των κληρονομημένων συναρτήσεων, εφαρμόζει κλήσεις εξουσιοδότησης μέσω αυτού του στιγμιότυπου.
  - Δηλ. κάθε επέκταση γίνεται αυτόματα adapter στην αυθεντική κλάσης

Decorator (4/7)

- **Επιπτώσεις**
  - Η decoration κλάση είναι πρακτικά η αυθεντική κλάση με δυνατότητα προσθήκης επιπλέον χαρακτηριστικών κατά την εκτέλεση σε όποιο στιγμιότυπο κληρονομεί από την αυθεντική κλάση
  - Ο συνδυασμός των decorators είναι εφικτός με τρόπο ανάλογο της *σύνθεσης συναρτήσεων*.



Decorator (5/7)

παράδειγμα (1/3)

```
class Car {
   public:
   virtual void Fly (void)=0 {}
   virtual void Move (void)=0 {}
   virtual void SpeedUp (void)=0 {}
};
classs Basic_Car : public Car
  { <implements all methods, some empty> };
class ABS_Car : public Car {
   private:
   Car* carInst;
   public:
   virtual void Move (void) override { carInst->Move(); }
   virtual void Break (void) override;
   ABS_Car (Car* car) : carInst (car) {...}
};
class TurboEngine_Car : public Car {
   public:
   TurboEngine_Car (Car* car) : carInst (car) {...}
   virtual void SpeedUp (void) override;
};
```

Decorated delegation instance

Υποχρεωτική υλοποίηση μέσω απλού delegation call όσων methods δεν αλλάζουν

Υλοποίηση όσων μεθόδων εξειδικεύονται ουσιαστικά

Decorator (6/7)

παράδειγμα (2/3)

```
class TractionControl_Car   : public Car {...};
class WD4_Car               : public Car {...};

Car* myCar          = new WD4_Car(new TurboEngine_Car(new Basic_Car));
Car* addTraction    = new TractionControl_Car(myCar);
```

New **Θα πρέπει να σημειωθεί ότι ενώ μία decorator κλάση κληρονομεί από την αυθεντική κλάση, και κατά την εκτέλεση συμπεριφέρεται σαν το στιγμιότυπο το οποίο «κοσμεί» με επιπλέον λειτουργίες, είναι τελείως διαφορετικός δείκτης από ότι το «κοσμούμενο» στιγμιότυπο (δηλ. δεν λειτουργεί το up-casting). Ωστόσο, μπορούμε να βάλουμε ειδική συνάρτηση που να επιστρέφει το «κοσμούμενο» στιγμιότυπο.**

```
class BasicCar : public Car {
   virtual Car* Pure (void) { return this; }
};
class TractionControl_Car : public Car {
   Car* Pure (void) { return carInst->Pure(); }
};
```

Αυτή είναι αναδρομική κλήση μέχρι το πρώτο undecorated instance

Decorator (7/7)

παράδειγμα (3/3)

```
class TractionControl_Car : public Car {
   virtual Car* Pure (void);
   virtual Car* Undecorated (void) { return carInst; }
};
```

Επιστρέφει το προηγούμενο στιγμιότυπο χωρίς την προσθήκη (decoration) λειτουργιών του caller

Με την **Pure()** και **Undecorated()** μπορούμε τελείως ή μερικώς να αφαιρέσουμε τη λειτουργική «διακόσμηση» που έχει προστεθεί στο αρχικό στιγμιότυπο. Αυτό είναι ειδικά χρήσιμο σε περιπτώσεις όπου:

ορισμένες προσθήκες λειτουργίας είναι παράνομο να εφαρμοστούν πάνω από άλλες προσθήκες (δηλ. θέλουμε το pure)

- πρέπει να αναιρούν πλήρως άλλες προσθήκες (εφαρμόζονται απ ευθείας στο pure)
- απλώς θέλουμε να κινηθούμε στα επίπεδα πρόσθετης λειτουργικότητας που έχουν εισαχθεί δυναμικά πάνω από ένα στιγμιότυπο

Αφαιρεί ένα το top decoration από ένα car instance

```
void Car::Pop (Car** car)
   { Car* prev = (*car)->Undecorated(); delete *car; *car = prev; }
```

Decorator - ένθετο

```
struct C {};

// mixin classes: generic derived class (base is a template parameter)
template <class T> struct C1 : public T {};
template <class T> struct C2 : public T {};
template <class T> struct C3 : public T {};


// C <- C1 <- C2 <- C3  (<- isa)

C3< // C2 <- C3
    C2< // C1 <- C2
        C1<C> // C <- C1
            >
                > c;

C2<C3<C2<C3<C> > > > c2;
C3<C2<C3<C2<C> > > > c3;

void f (C*) {}
void g (C2< C1<C> >*) {} // Error: we do not have something like C2 alone!

int main()
{
    f(&c2);
    f(&c);
    g(&c); // Ok, as C3 c inherits from that.
    g((C2< C1<C> >*) &c2); // Problem: the casting is unsafe:
    return 0;
}
```