CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2010
Doc 20 Type Object & Dependency Injection
20 Apr 2010

References

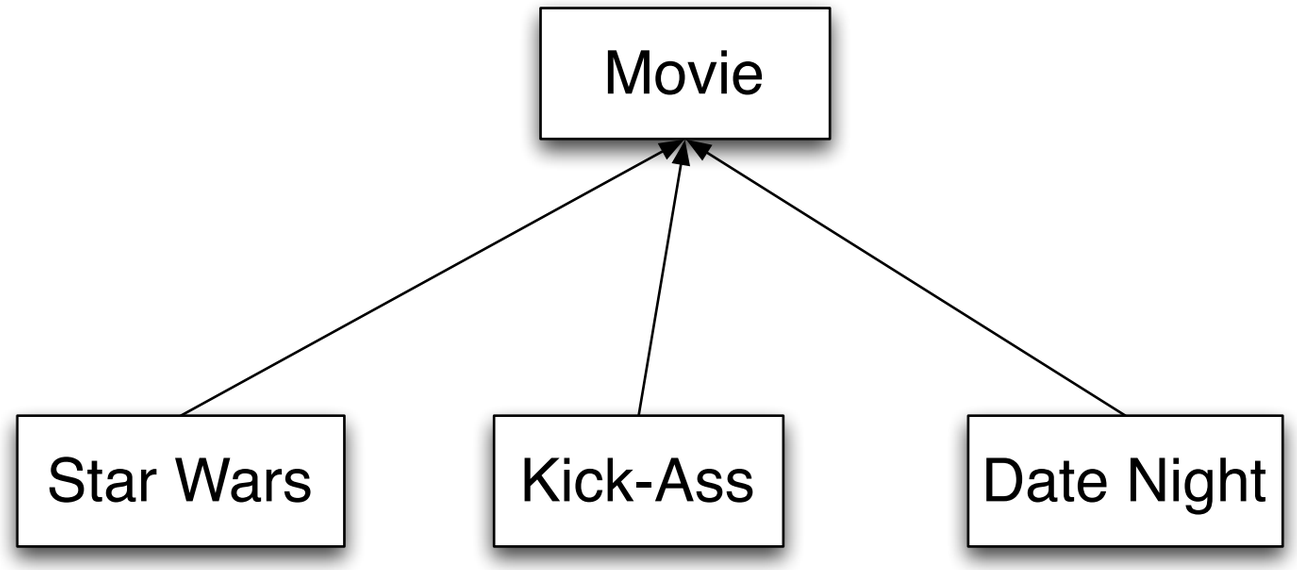
PicoContainer, http://www.picocontainer.org/

Type Object Pattern, http://www.ksc.com/article3.htm

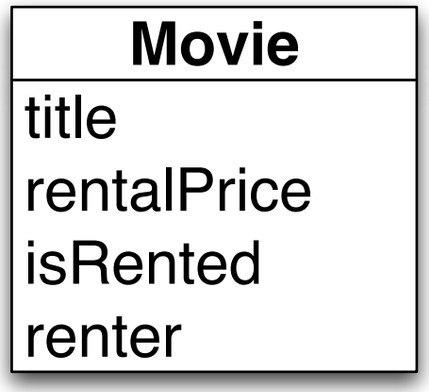
Inversion of Control Containers and the Dependency Injection pattern, http://martinfowler.com/articles/injection.html

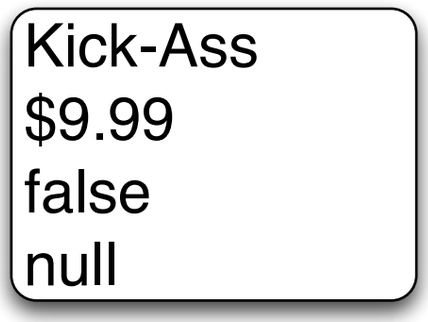
Type Object

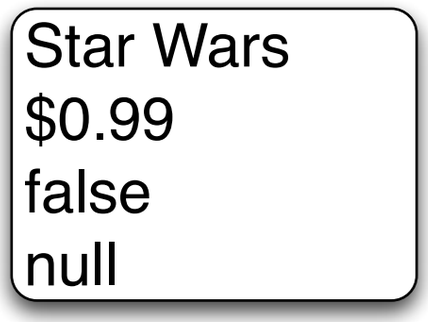
Motivation

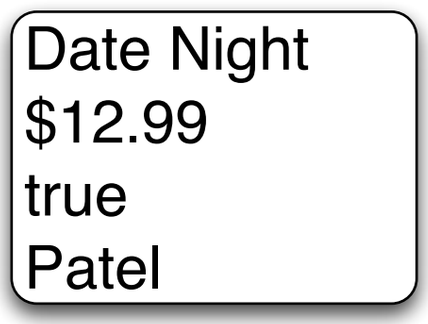


Why not use one class

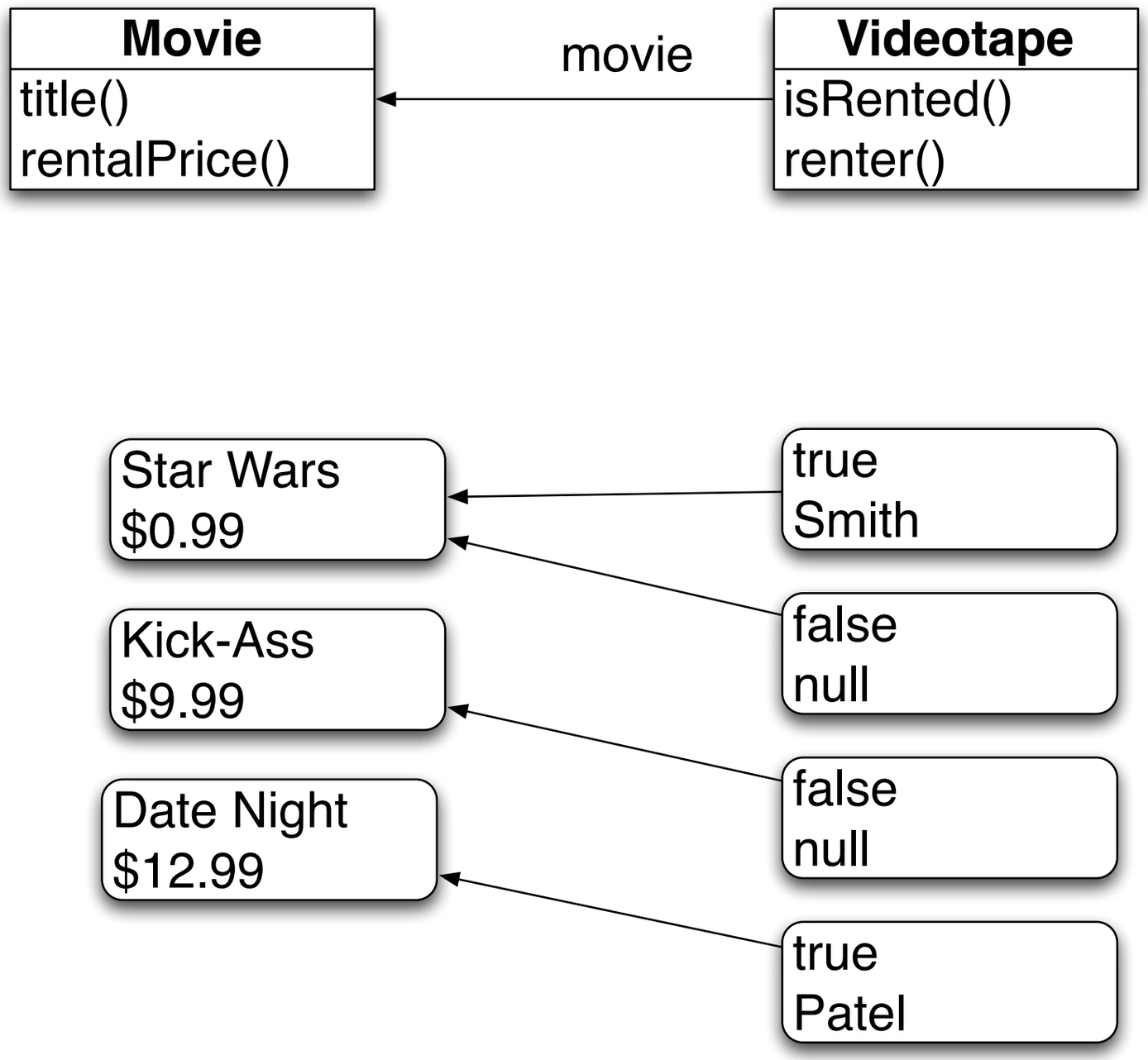




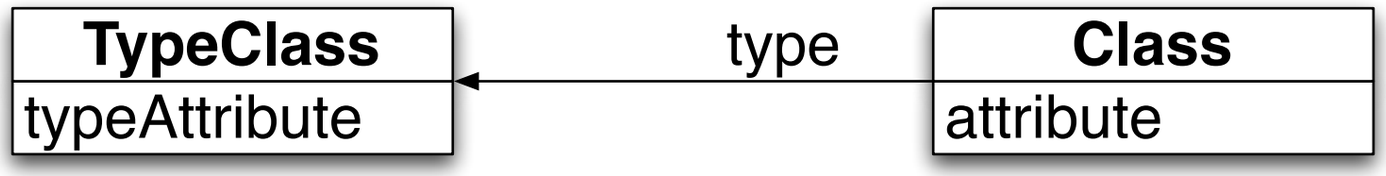




Type Object Solution



Structure



TypeClass

Class of TypeObject

TypeObject

Instance of TypeClass

Separate object for each type

All common properties of type

Class

Class of Object

Represents instances of TypeClass

Object

Instance of Class

Unique item with unique context

Applicability

Instances of a class need to be grouped together
according to their common attributes and/or behavior

Applicability

The class needs a subclass for each group to implement that group's common attributes and behavior.

Applicability

The class requires a large number of subclasses and/or the total variety of subclasses that may be required is unknown.

Applicability

You want to be able to create new groupings at runtime that were not predicted during design.

Applicability

You want to be able to change an object's subclass after its been instantiated without having to mutate it to a new class.

Applicability

You want to be able to nest groupings recursively so that a group is itself an item in another group.

Other Patterns

vs Strategy and State

vs Bridge

vs Decorator

vs Flyweight

Inversion of Control
Dependency Injection

Normal Control

```
class NormalControl {
    public void foo() {
        ArrayList example = new ArrayList();
        example.add( "cat");
    }

}
```

Inversion of Control - GUI

```
import  java.awt.*;
class  HelloApplication  extends  Frame
      {
      public  void  paint(  Graphics  display  )
            {
            int startX  =  30;
            int startY  =  40;
            display.drawString(  "Hello World", startX, startY  );
            }
}
```

Components & Services

What is the difference?

PicoContainer

highly embeddable, full-service, Inversion of Control (IoC) container
for components honor the Dependency Injection pattern

PicoContainer - Trivial Example

```
MutablePicoContainer pico = new DefaultPicoContainer();
pico.addComponent(ArrayList.class);
List list = (List) pico.getComponent(ArrayList.class);
```

```
List list = new ArrayList();
```


http://www.picocontainer.org/introduction.html

Juicer Example Classes

```
public interface Peelable {
    void peel();
}

public class Apple implements Peelable {
    public void peel() {
    }
}
```

```
public class Juicer {
    private final Peelable peelable;
    private final Peeler peeler;

    public Juicer(Peelable peelable, Peeler peeler) {
        this.peelable = peelable;
        this.peeler = peeler;
    }
}
```

```
public class Peeler implements Startable {
    private final Peelable peelable;

    public Peeler(Peelable peelable) {
        this.peelable = peelable;
    }

    public void start() {
        peelable.peel();
    }

    public void stop() {    }
}
```


http://www.picocontainer.org/introduction.html

Using the Container

```
MutablePicoContainer pico = new DefaultPicoContainer();
pico.addComponent(Apple.class);
pico.addComponent(Juicer.class);
pico.addComponent(Peeler.class);

Juicer juicer = (Juicer) pico.getComponent(Juicer.class);
```

```
Peelable peelable = new Apple();
Peeler peeler = new Peeler(peelable);
Juicer juicer = new Juicer(peelable, peeler);
return juicer;
```

MovieLister Example

```
class MovieLister {
    private MovieFinder finder;

    public MovieLister() {
        finder = new ColonDelimitedMovieFinder("movies1.txt");
    }

    public Movie[] moviesDirectedBy(String arg) {
        List allMovies = finder.findAll();
        for (Iterator it = allMovies.iterator(); it.hasNext();) {
            Movie movie = (Movie) it.next();
            if (!movie.getDirector().equals(arg)) it.remove();
        }
        return (Movie[]) allMovies.toArray(new Movie[allMovies.size()]);
    }
```

Constructor Injection

```
class ColonMovieFinder {
    public ColonMovieFinder(String filename) {
        this.filename = filename;
    }
}
```

```
class MovieLister {
    private MovieFinder finder;

    public MovieLister(MovieFinder finder) {
        this.finder = finder;
    }

    public Movie[] moviesDirectedBy(String arg) {
       same as before;
    }
```

Setter Injection

```
class MovieLister {
    private MovieFinder finder;

    public void setFinder(MovieFinder finder) {
        this.finder = finder;
    }
etc.
```