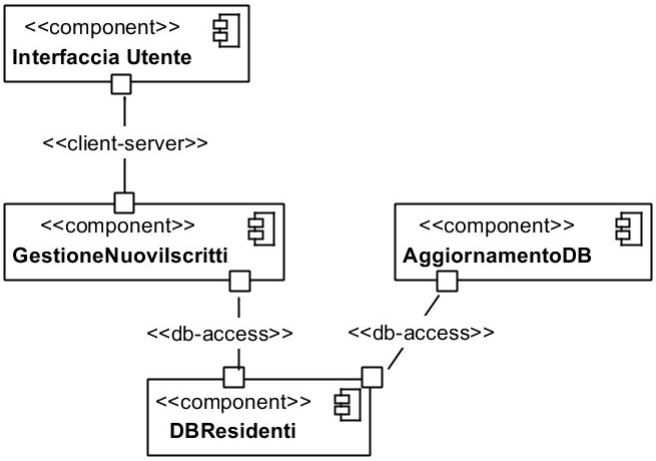
Esercitazione: Progettazione di dettaglio

Roberta Gori, Laura Semini
Ingegneria del Software
Dipartimento di Informatica
Università di Pisa

Homework

- *Con riferimento al caso di studio Pisa Mover si consideri la nuova tariffa scontata per i biglietti stazione ferroviaria-aeroporto: dal primo di dicembre 2017 è possibile, per ogni residente del comune di Pisa, acquistare fino a 6 biglietti al mese al costo di 1,20€ cadauno anziché 2,70€. Al momento l'acquisto è possibile solo presso la biglietteria aziendale situata nel parcheggio Via Aurelia, presentando la carta di identità, ma si desidera modificare le biglietterie automatiche in modo da permettere la vendita di biglietti scontati. L'utente interessato deve registrarsi presso la biglietteria aziendale o via web, fornendo generalità, comune di residenza e codice fiscale. Il sistema controlla la veridicità della dichiarazione usando i dati mantenuti in locale, e in caso di verifica positiva il suo codice fiscale viene inserito tra quelli accettati dalle biglietterie automatiche. In questo modo l'utente potrà acquistare biglietti a prezzo scontato presso le biglietterie automatiche, indicando il codice fiscale. Ogni tre mesi il sistema richiede la lista dei residenti del comune di Pisa, con una richiesta a un servizio offerto dall'Anagrafe, e aggiorna il proprio database: elenco dei residenti da consultare per future richieste di registrazione; lista dei codici fiscali accettati dalle biglietterie automatiche, (rimuovendo i non più residenti).*
- **Si** disegni un diagramma UML ibrido, C&C/deployment, in cui si evidenziano i principali componenti software e hardware necessari per realizzare il nuovo requisito relativo alla gestione dei biglietti scontati.
- Si dia un diagramma di struttura composita della componente, tra quelle definite nell'esercizio 3, che comunica con l'Anagrafe.

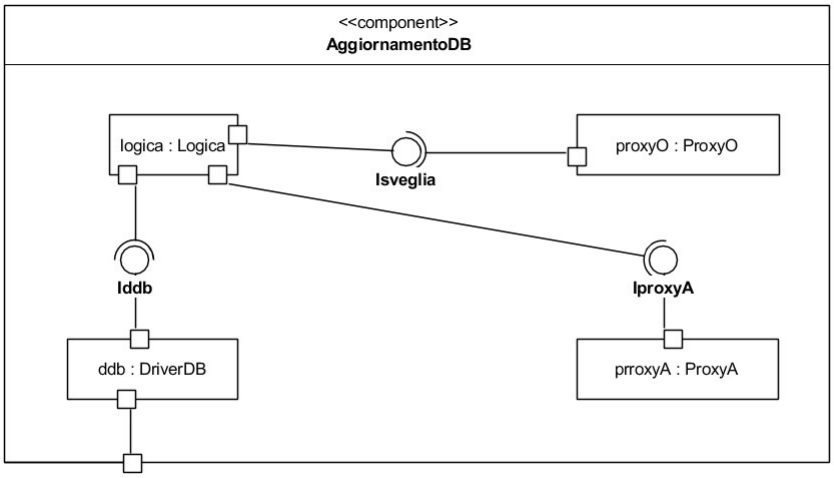
C&C



Deployment

- Da fare per esercizio, dislocando
  - l' artefatti che manifesta l'interfaccia utente su una macchine client (ci sono molte macchine client),
  - e gli artefatti che manifestano le altre componenti su una macchine server

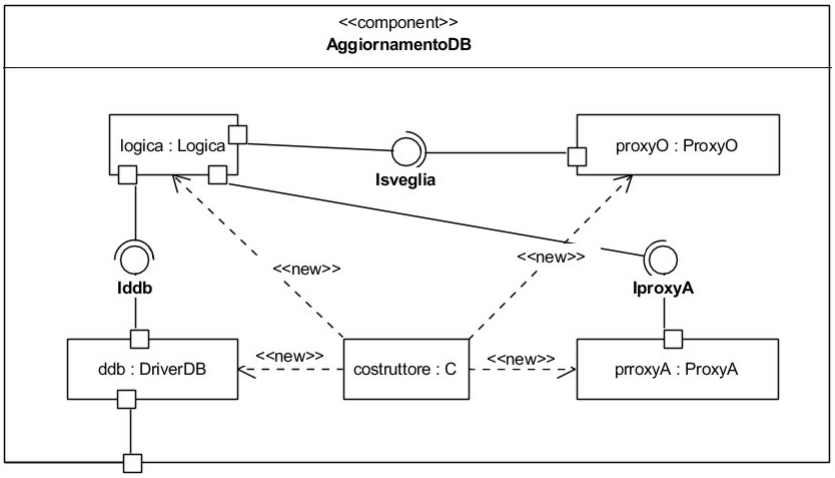
Struttura composita



Estensione

- Supponiamo ora di voler definire una parte che costruisce la componente

Struttura composita con creatore della componente



Comportamento del costrutture

- Prima costruisce ddb e proxyA
- poi la logica, passando ddb e proxyA come riferimenti
- poi proxyO, passando logica come riferimento.

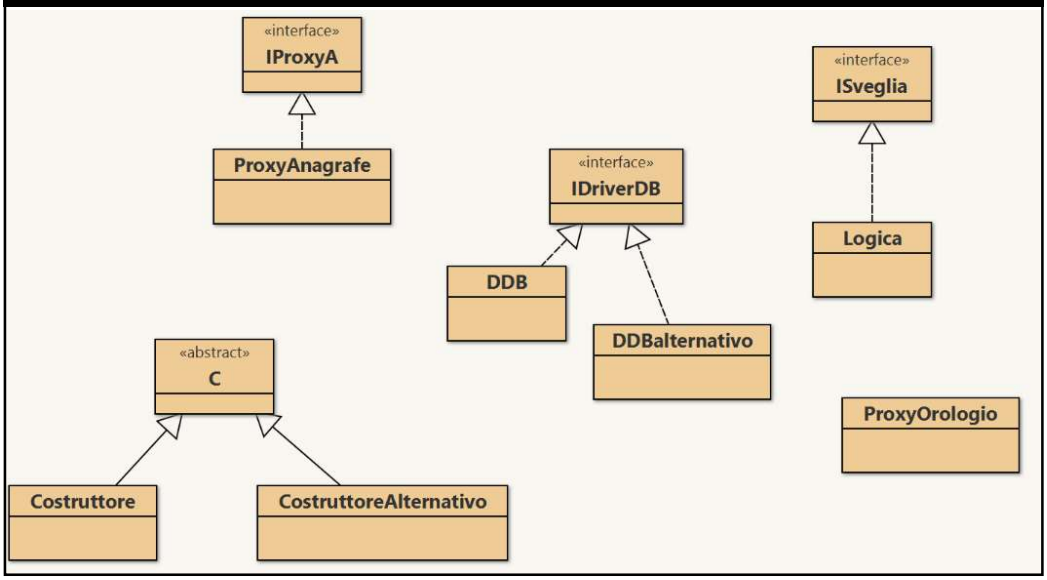
- Si documenta in modo naturale con un diagramma di sequenza (fare per esercizio)

Applicate Factory Method per il costruttore

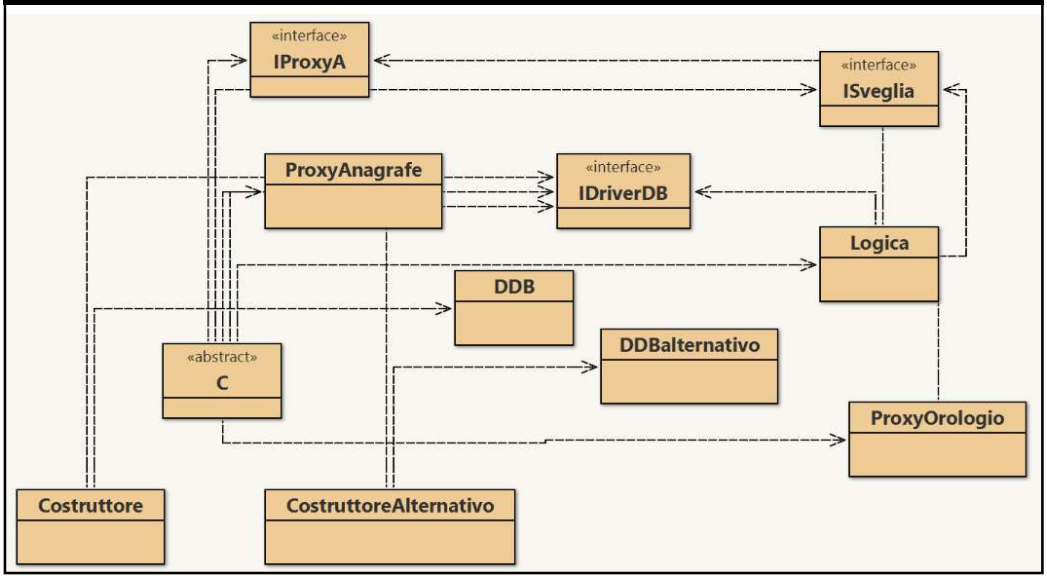
- Assumiamo ci siano due driver per il DB, alternativi uno all'altro (DBB e DBBalternativo) per interagire con DB realizzati in modo (e.g. linguaggio) diverso
- Usare FactoryMethod per definire il costruttore
  - Ci saranno
    - Un costruttore astratto che definisce la sequenza di passi (del lucido precedente) e che lascia astratta la creazione di oggetti di tipo IDriverDB

      protected abstract IDriverDB creaDDB();
    - Due costruttori concreti che implementano il metodo creaDDB()
      1. protected IDriverDB creaDDB(){ return new DDB(); }
      2. protected IDriverDB creaDDB(){ return new DDBalternativo(); }

Vista Strutturale di generalizzazione



Vista Strutturale d'uso



Creator del pattern

```
public abstract class C
{
  protected abstract IDriverDB creaDDB();

  public void MakeGestioneAgg()
   {
      IDriverDB ddb = creaDDB();
      IProxyA ipa = new ProxyAnagrafe();
      ISveglia logica = new Logica(ddb,ipa);
      ProxyOrologio po = new ProxyOrologio(logica);
   }
}
```

Concrete creators

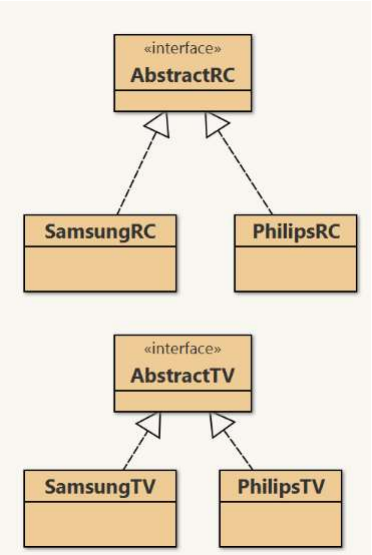
```
public class Costruttore extends C {
   protected IDriverDB creaDDB(){
      return new DDB();
   }
}
public class CostruttoreAlternativo extends C {
   protected IDriverDB creaDDB(){
      return new DDBalternativo();
   }
}
```

Homework

- Apply the factory patterns to produce:
  - Products: TVs and Remote controls (RC)
  - Of two brands: Samsung and Philips

1. With Concrete Factory: using parameters.
2. With Factory method: creator builds a TV and its RC, then packs them togheter.
3. With Abstract Factory: a client chooses the factory and asks for the product(s) he needs, for instance one TV and two remote controllers.

I prodotti (immutati in tutte le soluzioni per la factory) vista strutturale di generalizzazione

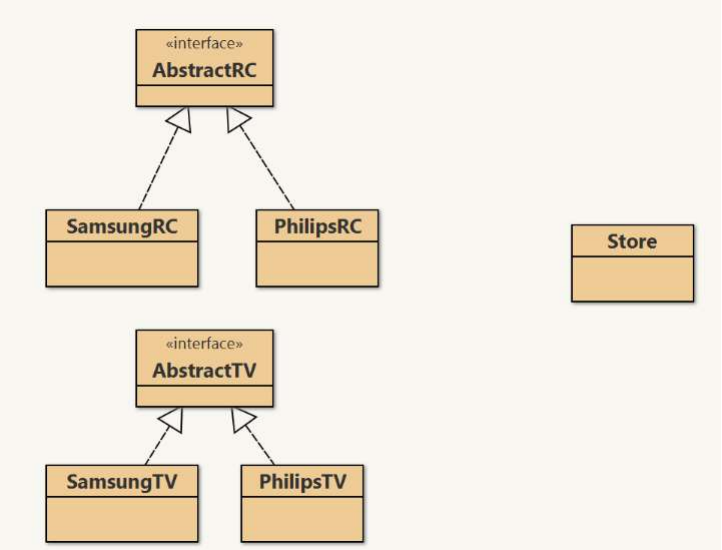


I prodotti: l'interfaccia TV e le tv concrete (RC analogo)

L'ESEMPIO SERVE SOLO PER SPIEGARE IL PATTERN, IL CODICE DI QUESTE CLASSI E' MINIMALE

```
public interface AbstractTV {
    public void getDescription();
}
-------
public class SamsungTV implements AbstractTV {
     public void getDescription() {
        System.out.println("Samsung TV");
    }
}
-------
public class PhilipsTV implements AbstractTV {
    public void getDescription() {
        System.out.println("Philips TV");
    }
}
```

Aggiungiamo una simple factory, la classe Store

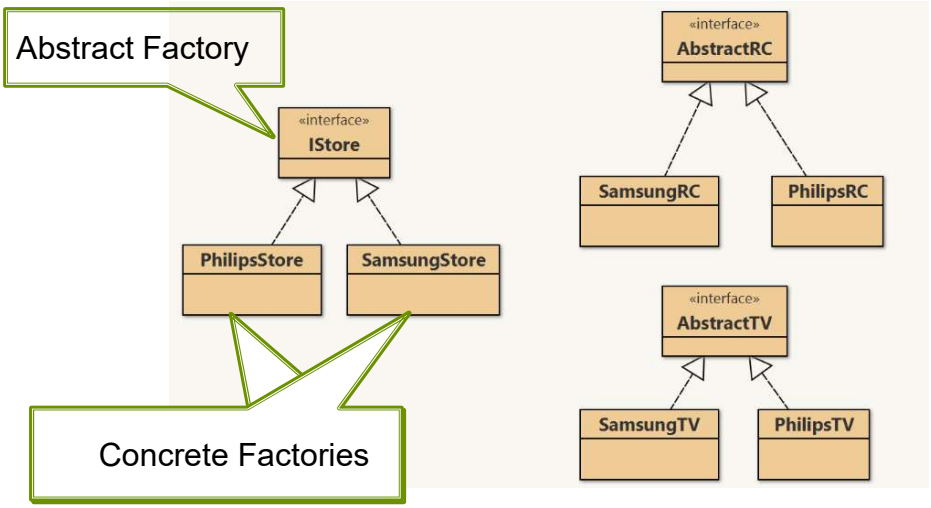


Codice di Store (con scelta minimalista, anche se poco felice, per la gestione del parametro)

```
public class Store{

   public AbstractTV makeTV(int code) {
      if (code == 1)
         return new SamsungTV();
         else
         return new SamsungTV();
   }
}
```

Applicando Abstract Factory



Il codice

```
public interface IStore {
   public AbstractTV makeTV();
   public AbstractRC makeRC();
}
```

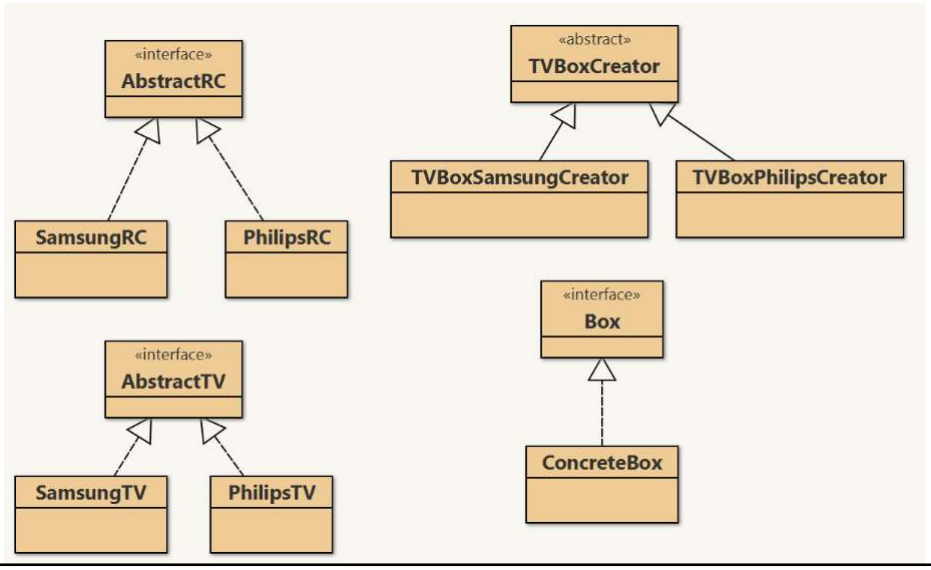
```
public class PhilipsStore implements IStore
{
   public AbstractTV makeTV() {
      return new PhilipsTV();
   }
    public AbstractRC makeRC() {
      return new PhilipsRC();
   }
}
```

```
public class SamsungStore implements IStore {
   public SamsungStore() {
   }
   public AbstractTV makeTV() {
      return new SamsungTV();
   }
   public AbstractRC makeRC() {
      return new SamsungRC();
   }
}
```

Applicazione di Factory Method

- Factory method risponde a esigenze particolari, in particolare avere un «algoritmo di costruzione» standard, con alcuni passi di creazione implementati nelle sottoclassi.
- In questo caso l'algoritmo è:
  1. costruire una scatola,
  2. costruire una TV,
  3. metterla nella scatola,
  4. costruire un RC,
  5. metterlo nella scatola

Applicazione di Factory method: Vista strutturale di generalizzazione



Interface Box

```
public interface Box
{
   public void addTV(AbstractTV tv);
   public void addRC(AbstractRC rc);
}
```

La classe "creator"

```
public abstract class TVBoxCreator
{
   public abstract AbstractTV creaTV();
   public abstract AbstractRC creaRC();

   public Box createTVBox()
   {
      Box b = new ConcreteBox();
      AbstractTV tv = creaTV();
      b.addTV(tv);
      AbstractRC rc = creaRC();
      b.addRC(rc);
      return b;
   }
}
```

I "creators" concreti

```
public class TVBoxSamsungCreator extends TVBoxCreator
{
   public AbstractTV creaTV(){return new SamsungTV();}
   public AbstractRC creaRC(){return new SamsungRC();}
}
--------
public class TVBoxPhilipsCreator extends TVBoxCreator
{
   public AbstractTV creaTV(){return new PhilipsTV();}
   public AbstractRC creaRC(){return new PhilipsRC();}
}
```