

Object Pool

- Un object pool è una deposito di istanze già create, una istanza sarà estratta dal pool quando un client ne fa richiesta
 - Il pool può crescere o può avere dimensioni fisse
 - Dimensioni fisse: se non ci sono oggetti disponibili al momento della richiesta, non ne creo di nuovi
 - Il client restituisce al pool l'istanza usata quando non più utile
- Il design pattern Factory Method può implementare un object pool
 - I client fanno richieste, come visto prima per il Factory Method
 - I client dovranno dire quando l'istanza non è più in uso, quindi riusabile
 - Lo stato dell'istanza da riusare potrebbe dover essere ri-scritto
 - L'object pool dovrebbe essere unico -> uso un Singleton

E. Tramontana - Pool, Dependency - 12 May 10 1

Esempio codice Object Pool

```
import java.util.LinkedList;
// CreatorPool è un ConcreteCreator e implementa un Object Pool
public class CreatorPool extends ShapeCreator {
    private LinkedList<Shape> pool = new LinkedList<Shape>();
    // getShape() è un metodo factory che ritorna un oggetto prelevato dal pool
    public Shape getShape() {
        Shape s;
        if (pool.size() > 0) s = pool.remove();
        else s = new Circle();
        return s;
    }
    // releaseShape() inserisce un oggetto nel pool
    public void releaseShape(Shape s) {
        pool.add(s);
    }
}
```

E. Tramontana - Pool, Dependency - 12 May 10 2

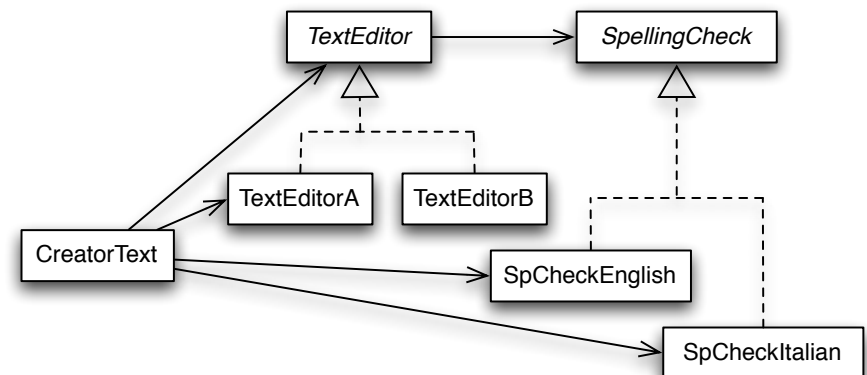
Dependency Injection

- Il design pattern Factory Method può essere usato per inserire le dipendenze necessarie ad altri oggetti (istanze di ConcreteProduct)
- Dependency injection
 - Una classe C usa un servizio S (ovvero C *dipende* da S)
 - Esistono tante implementazioni di S (ovvero S1, S2), la classe C non deve dipendere dalle implementazioni S1, S2
 - Al momento di creare l'istanza di C, indico all'istanza di C con quale implementazione di S deve operare
- Esempio di dependency
 - Una classe TextEditor usa un servizio SpellingCheck
 - Ci sono tante classi che *implementano* il servizio SpellingCheck, in base alla lingua usata: SpCheckEnglish, SpCheckItalian, etc.
 - TextEditor deve poter essere collegato ad una delle classi che implementano SpellingCheck

E. Tramontana - Pool, Dependency - 12 May 10 3

Diagramma UML

- Esempio di Dependency Injection



E. Tramontana - Pool, Dependency - 12 May 10 4

Esempio Codice Dependency Injection

```
public class TextEditorA implements TextEditor { //TextEditorA è un ConcreteProduct
    private SpellingCheck speller;
    public TextEditorA(SpellingCheck sp) { // inserisco la dipen. fornendo sp,
        speller = sp; // istanza del serv., al costruttore
    }
    public void put(String s) {
        if (speller.check(s)) ...
        else ...
    }
}

public class CreatorText { // CreatorText è un ConcreteCreator
    public static TextEditor getEnglishEditor() {
        return new TextEditorA(new SpCheckEnglish());
    }
    public static TextEditor getItalianEditor() {
        return new TextEditorB(new SpCheckItalian());
    }
}
```

E. Tramontana - Pool, Dependency - 12 May 10 5

Note sul codice

• Considerazioni

- TextEditorA e TextEditorB svolgono il ruolo di ConcreteProduct
- L'attributo speller di TextEditorA è inizializzato, attraverso la chiamata al costruttore, con l'opportuna istanza
- Il metodo factory getEnglishEditor() conosce la classe per lo spelling da usare ed inserisce (inject) la dipendenza sull'istanza di TextEditorA
- Un unico metodo factory istanzia e crea le dipendenze
 - Se usassi molteplici classi ConcreteCreator spargerei le decisioni di istanziazione
- All'interno della classe TextEditorA non è fissato l'uso di una classe che implementa SpellingCheck
- Posso sostituire la classe che implementa SpellingCheck facilmente
- Ho ottenuto la composizione di comportamenti piccoli e separati

E. Tramontana - Pool, Dependency - 12 May 10 6