Laboratorio di Sistemi Software

Progetto

Pattern Generator
Specifica iniziale

Luca Padovani (A-L)
Riccardo Solmi (M-Z)

Definizione del problema

- **Pattern Generator**
  - Libreria Java per definire dei pattern e, data una configurazione, generare il codice (Java) che li implementa.
- **Scopo del progetto**
  - Progettare ed implementare un generatore di codice
  - Fare pratica con i Design Pattern:
    - Progettazione della libreria
    - Definizione di generatori per un insieme di pattern

Definizione del Progetto

- **Progetto**
  - Ogni gruppo progetta e implementa la libreria e almeno 3 Design Patterns che la usano.
  - Un gruppo è formato da 3 studenti (anche misti AL/MZ). Sono ammessi, *solo in casi particolari e previo accordo con i docenti*, gruppi da 4 o 5 studenti che devono implementare rispettivamente 4/5 patterns e gruppi da 1 o 2 studenti che devono implementarne 2.
  - I Design Patterns vanno scelti tra quelli visti durante le lezioni (passate *e* future). Almeno 1 di ogni tipo (creazionale, strutturale, comportamentale).

Fasi del progetto

- **Prima Fase: Progettazione**
  - Progettazione della libreria con diagrammi UML e breve documento che illustra i trade-off e le scelte fatte.
  - Consegna obbligatoria
  - Vale fino a “moltissimi punti” (50%) se consegnata entro giovedì 8 maggio, non più di “pochissimi punti” (1) se consegnata dopo assieme al resto del progetto
- **Seconda fase: Discussione dei progetti in aula**
  - Discussione plenaria sulle scelte di design fino ad arrivare ad un progetto comune ragionevole da implementare (compresa definizione interfacce).
- **Terza fase: Implementazione**
  - Implementazione della libreria in Java e dei patterns scelti.

Generatori di codice

- **Consideriamo codice sintatticamente "parametrico"**
  - Si considera un pezzo di codice grezzo (una sequenza di caratteri) in cui sono presenti dei *buchi etichettati*
  - Codice grezzo + buchi etichettati = template
- **Istanziazione**
  - L'idea è di istanziare un template fornendo appositi riempimenti per i buchi
- **Composizione**
  - E' possibile riempire un buco con il risultato dell'istanziazione di un altro template

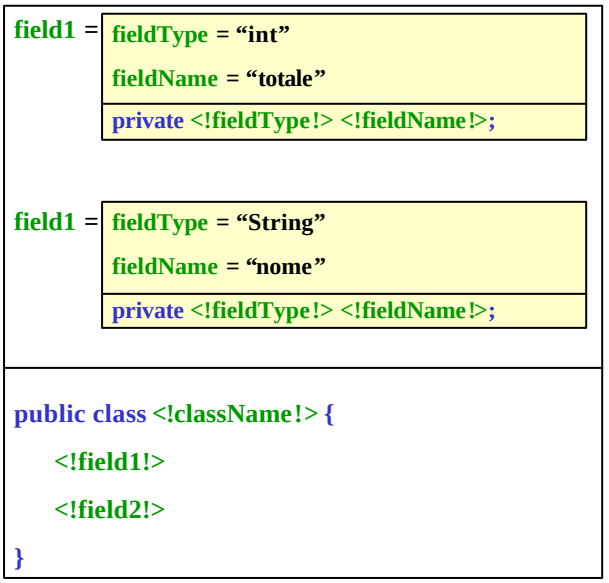
Esempio di generazione da template

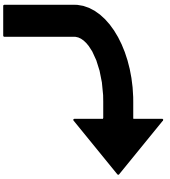
```
package <!packageName!>;
<!importList!>
public class <!className!> {
    public <!className!>() {
    }
    private int num<? = <!initVal!>?>;
}
```

```
package labss.prova;
import java.io.*;
import java.util.*;
public class TestClass {
    public TestClass() {
    }
    private int num;
}
```

- packageName = "labss.prova"
- importList = "import java.io.*;", "import java.util.*"
- className = "TestClass"

Esempio di generazione da frame strutturati





```
public class ? {
    private int totale;
    private String nome;
}
```


Fasi nella generazione di codice

1) Creazione di una configurazione di template (struttura ad albero) che rappresenta il codice che vogliamo produrre.

- La struttura può essere arbitrariamente complessa (DAG)
- Nella struttura posso definire istanziazioni parziali e/o totali per tutti/alcuni dei buchi dei template che compaiono in essa.
- I rimanenti buchi etichettati costituiscono i parametri del (albero di) template, che dovranno essere configurati opportunamente prima dell'istanziazione

2) Esportazione della struttura (costruita al punto 1)

- Il risultato dell'esportazione è una stringa di caratteri senza buchi

Code Section

- **Una Code Section è un template di codice che può contenere delle parti variabili e delle parti opzionali.**
- **Template di codice**
  - Codice sorgente
- **Parti variabili**
  - Riferimento ad uno slot, ovvero un “buco” nel codice sorgente ove verrà “inserito” il suo valore
- **Parti opzionali**
  - Sono Code Sections a loro volta
  - Vengono generate solo quando si verifica una determinata condizione (vedi oltre)

Esempio di Code Section

```
package <!packageName!>;
<!importList!>
public class <!className!> {
    public <!className!>() {
    }
    private int num<? = <!initVal!>?>;
}
```

LEGENDA:

Parti variabili: <!packageName!>, <!importList!>, <!className!>, <!initVal!>

Parti opzionali: <? = <!initVal!>?>;

Slot

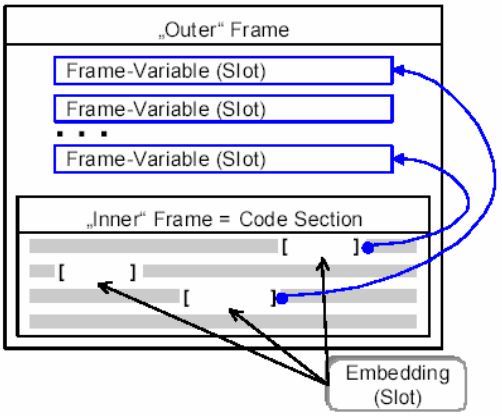
- **Uno slot è una variabile**
  - ha un nome ed eventualmente un valore ad esso associato
  - Uno slot senza valore associato è detto “non definito”
- **Uno slot non ha un “tipo”**
  - È possibile che in momenti diversi lo stesso Slot (con lo stesso nome) abbia valori di tipo diverso

Valore degli Slot

- **Ogni slot può assumere un valore a piacere dei seguenti tipi (variant):**
- **String**
  - Stringa costante di testo
- **Frame Reference**
  - Riferimento ad un Frame
- **Computed**
  - Metodo Java definito dal cliente
- **List**
  - Sequenza di valori “semplici” (gli altri tre tipi)

Frame

- **Un frame è una entità per generare del codice a partire da un template.**



Esempio:

fieldType = “String”

fieldName = “nome”

private <!fieldType!> <!fieldName!>;

Frame /2

- **Un Frame contiene un template di codice (Code Section), definisce il valore di un insieme di variabili (Slot)**
  - L’insieme degli Slot definiti in un Frame può essere diverso dall’insieme dei riferimenti a Slot contenuti nella sua Code Section (Frame aperto)
- **Il Frame rappresenta l’unità di scoping delle variabili che definisce (Slot)**
  - Le variabili sono visibili nella Code Section del Frame stesso e nei frame sottostanti (nella struttura ad albero)
- **Il Frame è l’unità principale di esportazione**
  - Il cliente della libreria creerà Frames ed esporterà Frames

Frame /3

- **I Frame saranno rappresentati da una classe Java opportuna che deve consentire:**
  - Di istanziare il Frame passandogli una Code Section sotto forma di stringa con la sintassi vista (es. “public <!className!>() { }”). NB. La Code Section non può essere cambiata in seguito
  - Di definire il valore di uno Slot attraverso dei metodi set e add. Una *set* assegna un valore “semplice” allo slot; una *add* fa diventare lo slot una lista e aggiunge un valore (“semplice”)
  - Di esportare il Frame, ovvero di generare il codice della sua Code Section

Generazione/esportazione di codice

- **Durante la fase di esportazione, la rappresentazione ad albero (grafo aciclico) di un Frame deve produrre una stringa**
- **Il metodo di esportazione deve accettare un argomento di tipo *PrintWriter* sul quale avviene mandato l’output prodotto**
- **E, data la presenza di identificatori (riferimenti a Slot), la generazione deve aver luogo in un *contesto* che definisce in un certo istante il valore di un insieme di Slot**
- **All’inizio della generazione il contesto è vuoto**

Esportazione di una Code Section

- **L'esportazione di una Code Section in un contesto C produce il testo che compare nella Code Section stessa in cui:**
  - Al posto di un "buco", cioè di un riferimento a Slot, viene esportato il valore dello Slot stesso (valutato in C)
  - Un blocco opzionale viene esportato solo se tutti gli Slot che contiene sono definiti (in C).

Esportazione di uno Slot

- **L'esportazione di uno Slot definito in un contesto C dipende dal tipo del suo valore:**
  - Un valore stringa produce la stringa stessa
  - Un valore Frame produce l'esportazione del Frame nel contesto C
  - Un valore calcolato esegue il codice fornito dal cliente, al quale viene passato il contesto C e il PrintWriter
  - Una sequenza di valori produce l'esportazione dei singoli valori, nell'ordine in cui compaiono
- **L'esportazione di uno Slot non definito è la stringa vuota**

Esportazione di un Frame

- **L'esportazione di un Frame in un contesto C produce l'esportazione della sua Code Section in un contesto C' tale che**
  - Gli Slot definiti dal Frame hanno in C' il (nuovo) valore ad essi associato
  - Gli Slot non definiti dal Frame e che erano definiti in C mantengono lo stesso valore che avevano in C
  - Gli Slot non definiti dal Frame e che non erano definiti in C restano non definiti anche in C'
- **NB Sono le stesse regole di scoping di un comune linguaggio strutturato (es. Java)**

Registro dei Frame

- **Deve essere possibile associare un nome ai Frame che vengono definiti in modo da renderli riusabili**
- **Per questo motivo introduciamo un Frame Registry, che consente di gestire un insieme di Frame:**
  - Inserire una nuova associazione nome/Frame
  - Recuperare il Frame associato ad un nome
    - NB Il Frame recuperato sarà una copia (un *clone*) del Frame nel registro, in modo che sia possibile istanziarlo più volte

Vincoli di compatibilità tra gruppi

- **Progettate il registro in modo che possa contenere Frame definiti da gruppi diversi.**
- **La generazione di codice deve funzionare anche in presenza di parti (Frame o Slot) scritte da altri gruppi.**
- **NB. Nel Documento consegnato nella fase 1 deve essere precisato come si pensa di rispettare questi vincoli.**
  - Esplicitare quali interfacce/classi devono essere comuni, quali invece sono specifiche per la vostra implementazione