

Laboratorio di Progettazione di Sistemi Software

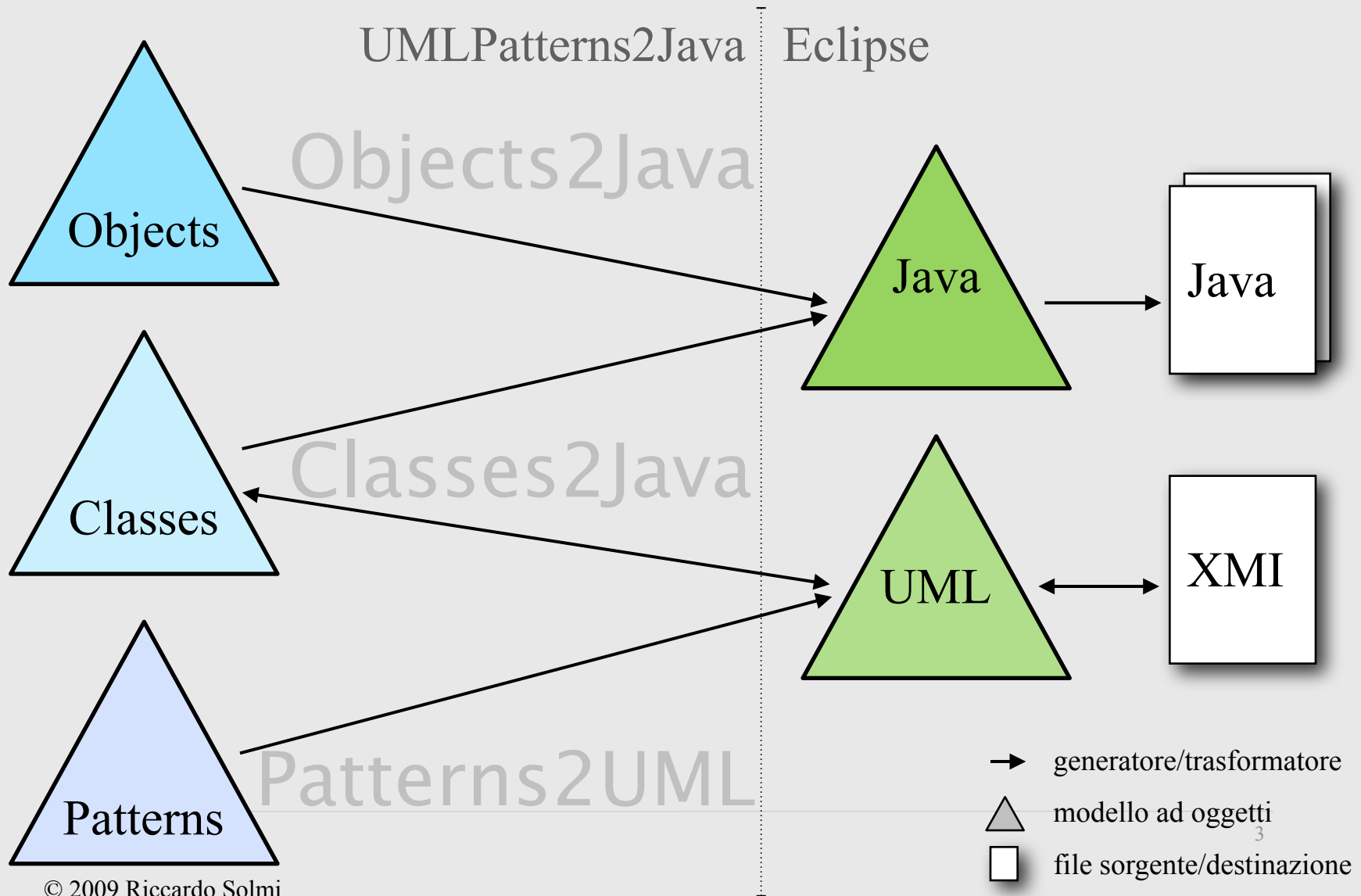
Progetto: UMLPatterns2Java

Riccardo Solmi

Progetto UMLPatterns2Java

- Progettare ed implementare uno strumento generativo che velocizzi l'applicazione programmatica di Design Patterns e Diagrammi UML in Java
- Il progetto si compone di tre sotto-progetti; ogni gruppo deve sceglierne uno:
 - UMLObjects2Java - Automazione per Diagramma degli Oggetti
 - UMLClasses2Java - Automazione per Diagramma delle Classi
 - Patterns2UMLClasses - Automazione per Design Patterns
- Il progetto si appoggia su due librerie open source di Eclipse che facilitano l'uso programmatico di Java e UML
 - I gruppi possono collaborare nella progettazione ed implementazione delle API creazionali comuni ai tre sotto-progetti

Scenari d'uso dello strumento



Sotto-progetti (uno a scelta per gruppo)

- **UMLObjects2Java**
 - Definire un modello semplificato del contenuto di un UML Object Diagram: Objects
 - Introdurre un pattern creazionale per il modello Java di Eclipse
 - Definire un generatore da modello Objects a modello Java con almeno due strategie
- **UMLClasses2Java**
 - Definire un modello semplificato del contenuto di un UML Class Diagram: Classes
 - Introdurre un pattern creazionale per il modello Java di Eclipse
 - Introdurre un pattern creazionale per il modello UML2 di Eclipse
 - Definire un generatore da modello Classes a modello Java
 - Definire due trasformatori da modello Classes a modello UML e viceversa
- **Patterns2UMLClasses**
 - Definire un modello per descrivere un Design Pattern a scelta: Patterns
 - Introdurre un pattern creazionale per il modello UML di Eclipse
 - Definire un generatore da modello Patterns a modello UML

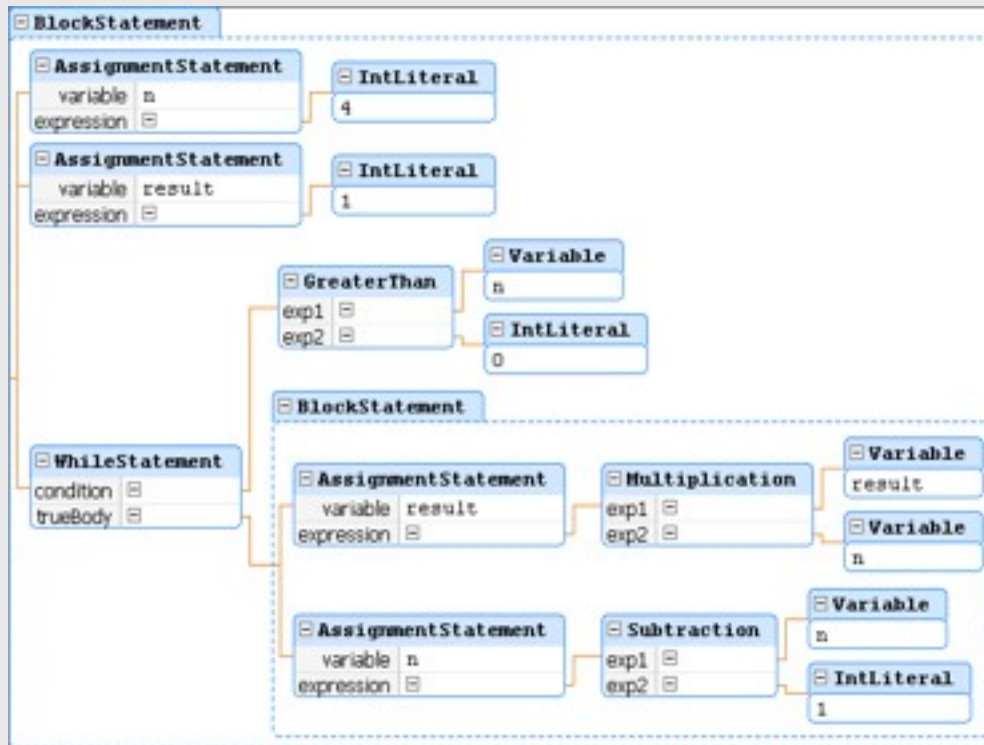
Precisazioni comuni

- I modelli Objects, Classes e Patterns si possono definire facendo uso di un diagramma delle classi UML.
- Le istanze di Objects, Classes e Patterns sono a loro volta dei modelli e sono definite all'interno di codice Java facendo uso di un pattern creazionale.
- I pattern creazionali da introdurre nei 3 sotto-progetti hanno lo scopo di disaccoppiare e semplificare l'uso programmatico dei linguaggi UML e Java.
- Per generatori e trasformatori si intende operazioni polimorfe che prendono in input un modello e producono in output un modello.
- E' a discrezione dei singoli gruppi "completare" i generatori/trasformatori con delle invocazioni ai servizi di persistenza testuale Java/XMI forniti dalle librerie Java e UML.

Precisazioni su Objects2Java

- Il generatore Objects2Java deve prendere in input un modello di Objects e produrre in output un modello Java corrispondente al codice che serve per costruire la struttura di oggetti rappresentata nel modello Objects di input.
- Il generatore deve sapere generare un modello Java con almeno due strategie diverse selezionabili dai clienti del generatore. Esempi di strategie:
 - invocazione di costruttori,
 - invocazione di metodi di una abstract factory,
 - invocazione di metodi di una factory statica,
 - invocazione di costruttori/metodi factory senza parametri con successive invocazioni di metodi setter, ...

Esempio di input di Objects2Java



- NB. L'input non è il diagramma disegnato ma il codice Java che, facendo uso di un pattern creazionale, costruisce in memoria una istanza del vostro modello Objects. Quindi l'input sarà del tipo:
 - `createObject("BlockStatement", createObject(...`

Esempi di due strategie di output di Objects2Java

```
public static Statement creaProgrammaFattorialeInIL() {
    return new Block(
        new Assignment(new Variable("n"), new Int(4)),
        new Assignment(new Variable("result"), new Int(1)),
        new While(new GreaterThan(new Variable("n"), new Int(0)), new Block(
            new Assignment(new Variable("result"),
                new Multiplication(new Variable("result"), new Variable("n"))),
            new Assignment(new Variable("n"),
                new Subtraction(new Variable("n"), new Int(1)))
        ));
}
```

```
public static Statement creaProgrammaFattorialeInILConFactory(ILAbstractFactory f) {
    return f.createBlock(
        f.createAssignment("n", f.createInt(4)),
        f.createAssignment("result", f.createInt(1)),
        f.createWhile(f.createGreaterThan(f.createVariable("n"), f.createInt(0)), f.createBlock(
            f.createAssignment("result",
                f.createMultiplication(f.createVariable("result"), f.createVariable("n"))),
            f.createAssignment("n",
                f.createSubtraction(f.createVariable("n"), f.createInt(1)))
        ));
}
```

NB. dovete generare i modelli Java di questi esempi non il testo!

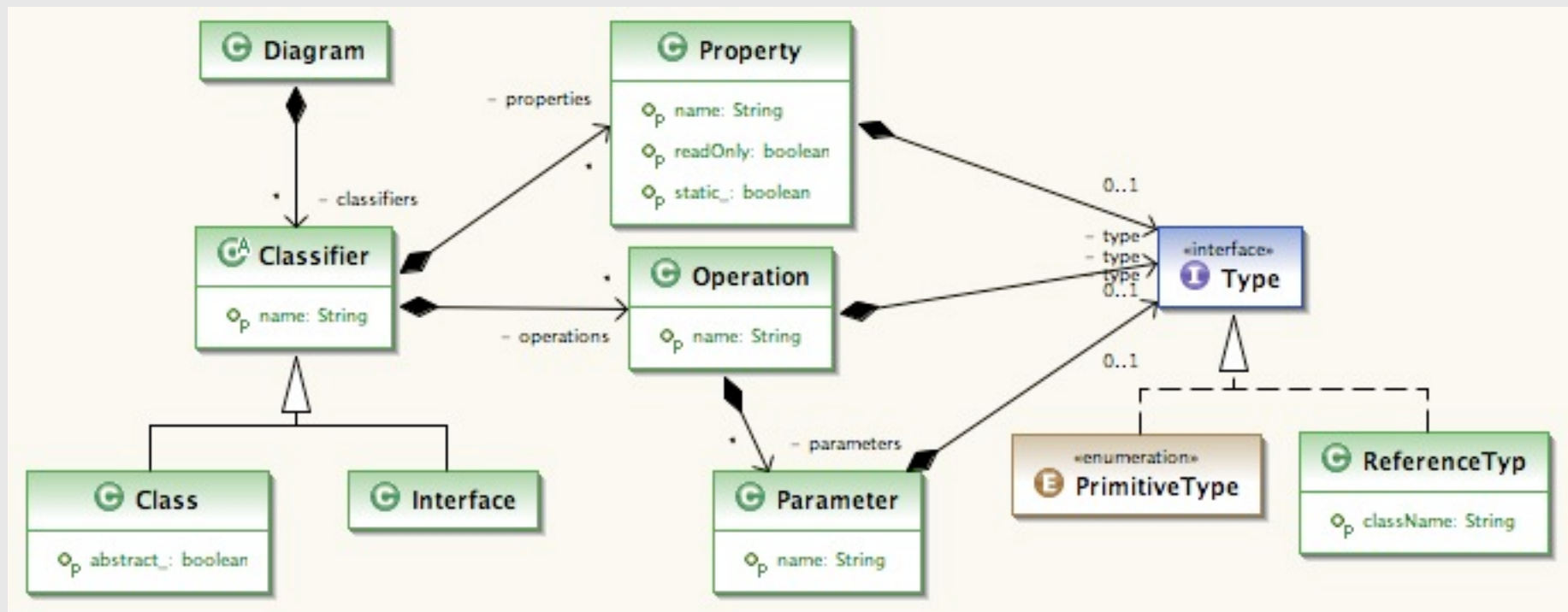
- ... usando le API creazionali introdotte per il modello Java di Eclipse.

Precisazioni su Classes2Java

- Il generatore Classes2Java deve produrre (i modelli delle) classi/interfacce Java descritti nel modello Classes ricevuto in input.
 - deve usare le API creazionali introdotte per il modello Java di Eclipse.
- E' a discrezione dei gruppi decidere il contenuto dei modelli generati; si può prendere spunto da quello che genera lo strumento UML che usiamo a lezione e generare di più o di meno. Possibili aggiunte:
 - costruttori con o senza parametri
 - generazione di una abstract factory
 - interfaccia comune astratta con supporto di Visitor e/o di API di manipolazione generica
 - getter e setter con stili diversi da quello ufficiale “java beans”

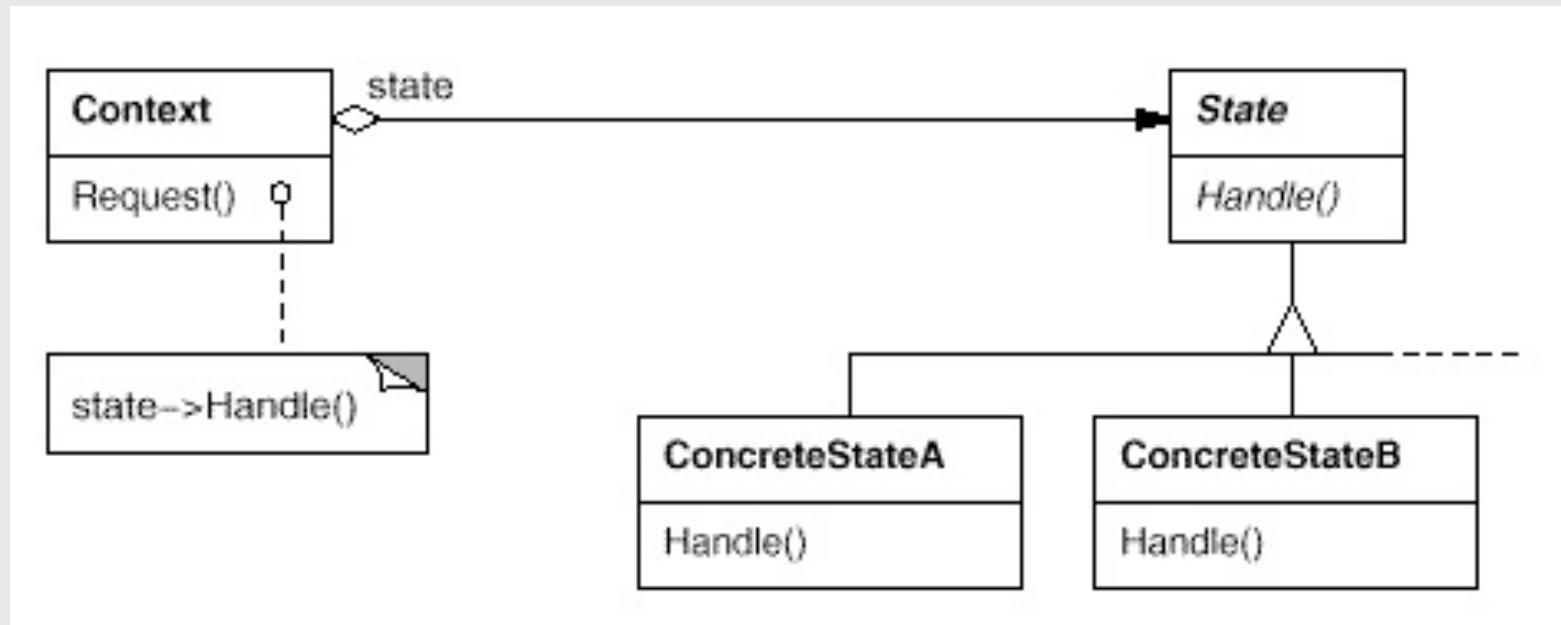
Esempio di modello per Classes

- Come base da cui partire si può lavorare sul diagramma riportato qua sotto
- Come punto di riferimento superiore si può prendere il modello ufficiale della libreria UML2



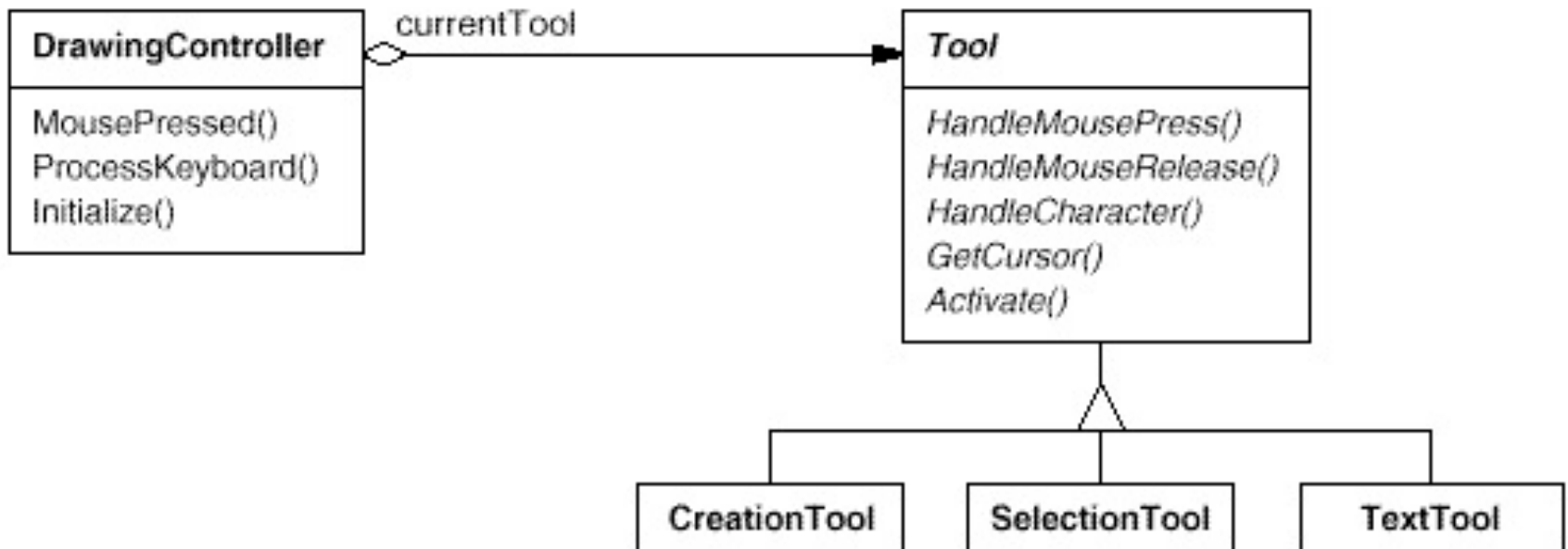
Precisazioni su Patterns2UMLClasses

- Poniamo che si scelga il design pattern State
 - si deve studiare sul catalogo il pattern (in particolare la sua struttura e i partecipanti)
 - aiutandosi guardando gli esempi, ci si deve chiedere che cosa si può/si deve scegliere e configurare per applicare il pattern ad un caso concreto

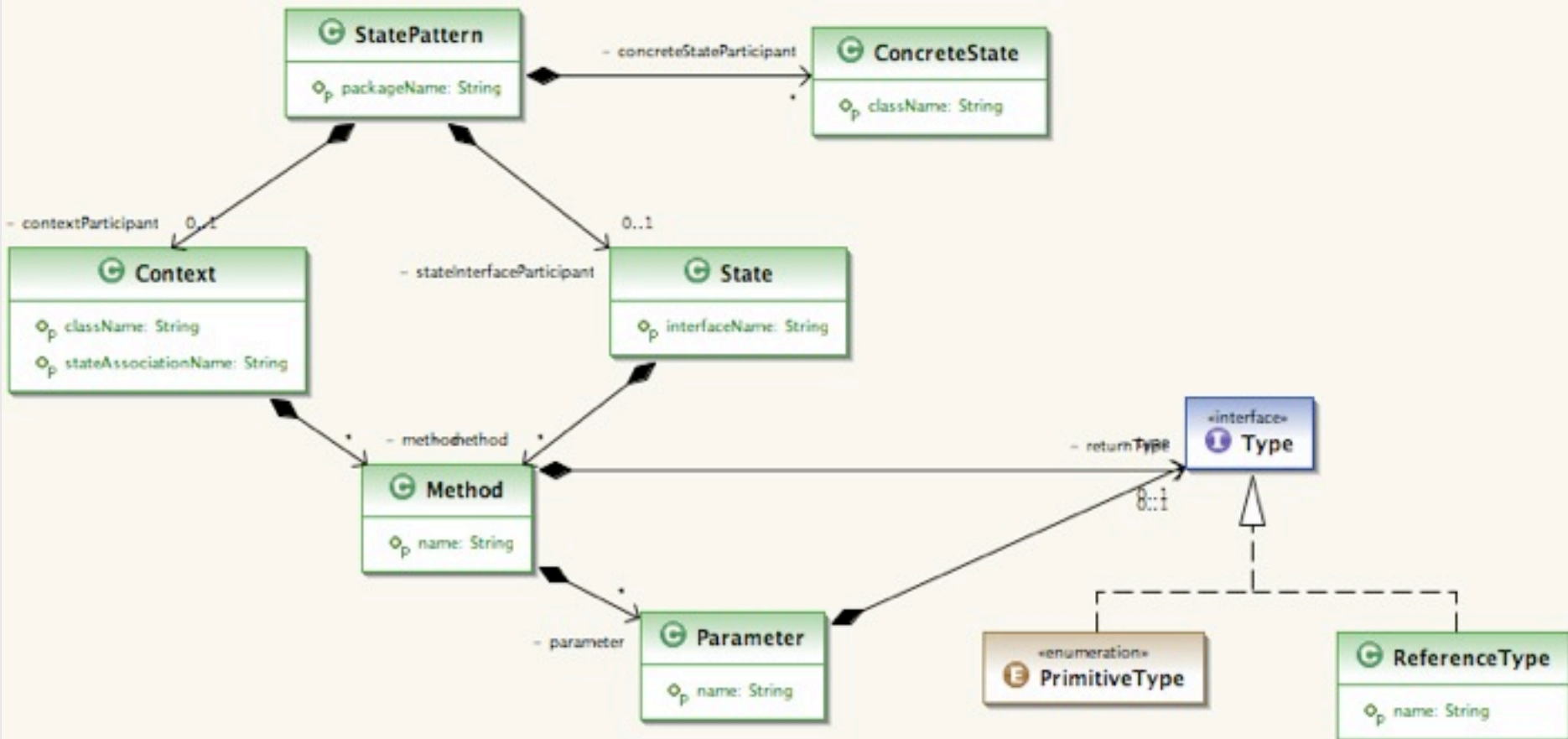


Esempio di applicazione del pattern State

- La struttura (diagramma delle classi) di un esempio di applicazione del pattern State è allo stesso tempo:
 - una guida per definire il modello Patterns di State
 - una guida per definire l'istanza del modello di Patterns corrispondente a questo esempio di applicazione del pattern
 - il diagramma dell'output che si vuole generare



Esempio di modello Patterns per il pattern State



Costruzione con Patterns dell'esempio

```
public static StatePattern createDrawingToolExample(StatePatternFactory f) {  
    return f.createStatePattern(  
        "labss.example.drawingtool",  
        f.createContext(  
            "DrawingController",  
            "currentTool",  
            f.createMethod("mousePressed"),  
            f.createMethod("processKeyboard"),  
            f.createMethod("initialize")),  
        f.createState(  
            "Tool",  
            f.createMethod("handleMousePress"),  
            f.createMethod("handleMouseRelease"),  
            f.createMethod("handleCharacter"),  
            f.createMethod("getCursor"),  
            f.createMethod("activate")),  
        f.createConcreteState("CreationTool"),  
        f.createConcreteState("SelectionTool"),  
        f.createConcreteState("TextTool"));  
}
```

Eclipse UML2

- **UML2 è l'implementazione ufficiale del modello UML 2.x per Eclipse**
 - È rivolta agli sviluppatori di strumenti UML
 - Si usa in modo programmatico
 - Facilita l'implementazione e l'interoperabilità degli strumenti UML
- **Servizi interessanti per il progetto:**
 - API creazionali per costruire modelli di classi
 - API per caricare/salvare i modelli UML nel formato XMI
- **Materiale didattico**
 - Sito ufficiale del progetto:
www.eclipse.org/uml2/
 - Wiki del progetto:
<http://wiki.eclipse.org/MDT-UML2>
 - Tutorial: Getting Started with UML2:
http://www.eclipse.org/modeling/mdt/uml2/docs/articles/Getting_Started_with_UML2/article.html
 - Javadoc API del package: org.eclipse.uml2.uml
<http://help.eclipse.org/stable/index.jsp?topic=/org.eclipse.jdt.doc.isv/reference/api/org.eclipse/jdt/core/dom/package-summary.html>

Eclipse Java AST/DOM

- **Java AST/DOM è l'implementazione ufficiale del modello di Java per Eclipse**
 - È rivolta agli sviluppatori di strumenti Java
 - Si usa in modo programmatico
 - Facilita l'implementazione e la composizione di strumenti che manipolano Java (generatori, refactoring, analizzatori, ...)
- **Servizi interessanti per il progetto**
 - API creazionali per costruire modelli di codice Java
 - API per caricare/salvare i modelli Java nel formato testuale

Eclipse Java AST/DOM /2

- **Materiale didattico**

- Tutorial: Manipulating Java code (in Help/JDT Plug-in Dev.Guide/Prog.Guide/JDT Core/)

http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.jdt.doc.isv/guide/jdt_api_manip.htm

- Tutorial: Abstract Syntax Tree

http://www.eclipse.org/articles/article.php?file=Article-JavaCodeManipulation_AST/index.html

- Tutorial: Exploring Eclipse's ASTParser

<http://www.ibm.com/developerworks/opensource/library/os-ast/>

- Javadoc API del package: org.eclipse.jdt.core.dom

<http://help.eclipse.org/stable/index.jsp?topic=/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/package-summary.html>

Vincoli di progettazione

- I pattern creazionali introdotti per i modelli UML2 e Java AST/DOM devono essere limitati ai prodotti che servono al progetto.
- L'implementazione dei modelli Patterns, Classes e Objects deve fornire:
 - API creazionali
 - API di manipolazione specifica e/o generica
 - Supporto ai Patterns per la definizione modulare del comportamento
- La costruzione dei modelli deve fare uso delle API creazionali
- L'implementazione dei generatori/trasformatori deve essere separata dai modelli

Altri requisiti e vincoli

- Il design del Sistema Software deve fare un uso estensivo ma ragionato dei Design Patterns visti a lezione e deve essere documentato con diagrammi UML
 - Il design e l'implementazione devono essere *conformi* (nella struttura e nella terminologia) ai Design Patterns studiati.
- Ogni gruppo deve scegliere un *nomeGruppo* di fantasia o derivarlo concatenando i prefissi di 3 lettere dei cognomi dei membri del gruppo.
- I sorgenti del progetto devono essere organizzati in packages che abbiano un prefisso comune del tipo: *labss.nomegruppo*
- Ogni membro di un gruppo deve scrivere almeno un esempio d'uso del progetto consegnato dal gruppo a cui appartiene. Gli esempi devono avere:
 - il sorgente scritto in un package *labss.nomegruppo.examples*
 - una procedura di esecuzione semplice e documentata.

Consegna

- Esportare il progetto (Eclipse) in un archivio ZIP e spedire come allegato di una e-mail indirizzata a: solmi@cs.unibo.it.
 - Come oggetto della e-mail usare formula del tipo:
“Consegna progetto LABSS *nomeGruppo*”
 - Come nome dell’archivio allegato usare il nome del gruppo.
 - La e-mail deve essere spedita da un membro del gruppo.
- Le date di consegna sono pubblicate sul sito del corso.