

Laboratorio di Progettazione di Sistemi Software

**Materiale per il progetto
con esercizi**

Valentina Presutti (A-L)

Riccardo Solmi (M-Z)

Indice degli argomenti

- **Progetto labp2001 e refactoring in labss_il_model**
- **Astrazione operazioni binarie e literal**
- **Aggiunta modulare di costrutti**
 - Factory estendibili di esempi e prototipi
 - API specifiche di modellazione
- **Aggiunta modulare di operazioni**
 - Visitors, type switch, Iterators
 - API generiche di modellazione

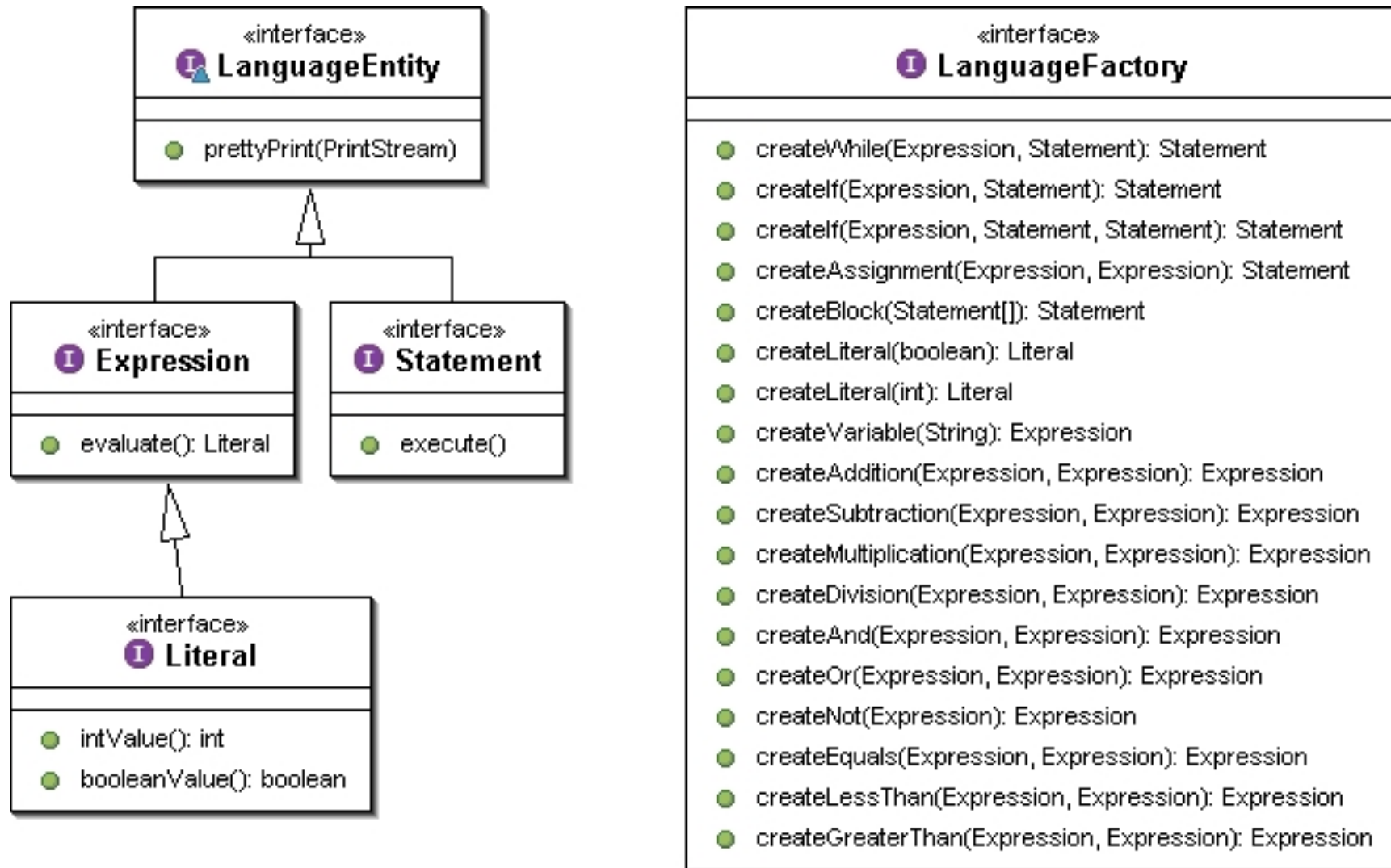
Introduzione

- **Questo documento contiene la descrizione del processo di sviluppo del progetto del corso di labss 2004/05**
- **Le scelte progettuali e le alternative sono presentate usando il catalogo di design pattern visto a lezione**
- **Le trasformazioni che dal progetto iniziale (primo esercizio) portano a sviluppare tutto il materiale per il progetto finale (progetto da svolgere) sono descritte usando il catalogo di refactoring presentato a lezione.**
- **Le domande e gli esercizi proposti vengono iniziati a lezione e sono da completare a casa.**

Progetto labp2001

- **Esercizio: implementare il progetto labp2001**
 - Importate il progetto vuoto dato (labp2001implvuota.zip)
 - Le specifiche sono incluse (labp2001implvuota/docs/api/index.html)
- **Usare il supporto di Eclipse per definire:**
 - le classi concrete, i metodi ereditati, i costruttori
- **Ad esempio si può partire dal main di Test:**
 - scommentare l'istanziamento della factory;
 - procedere guidati dalle lampadine sugli errori per creare le classi concrete;
 - usare il menu contestuale *source* per introdurre implementazioni di metodi ereditati e i costruttori usando i campi;
 - per implementare i metodi servirsi del menu contestuale che si apre quando si digita “oggetto punto” (esempio: exp1.)

Diagramma delle classi delle specifiche

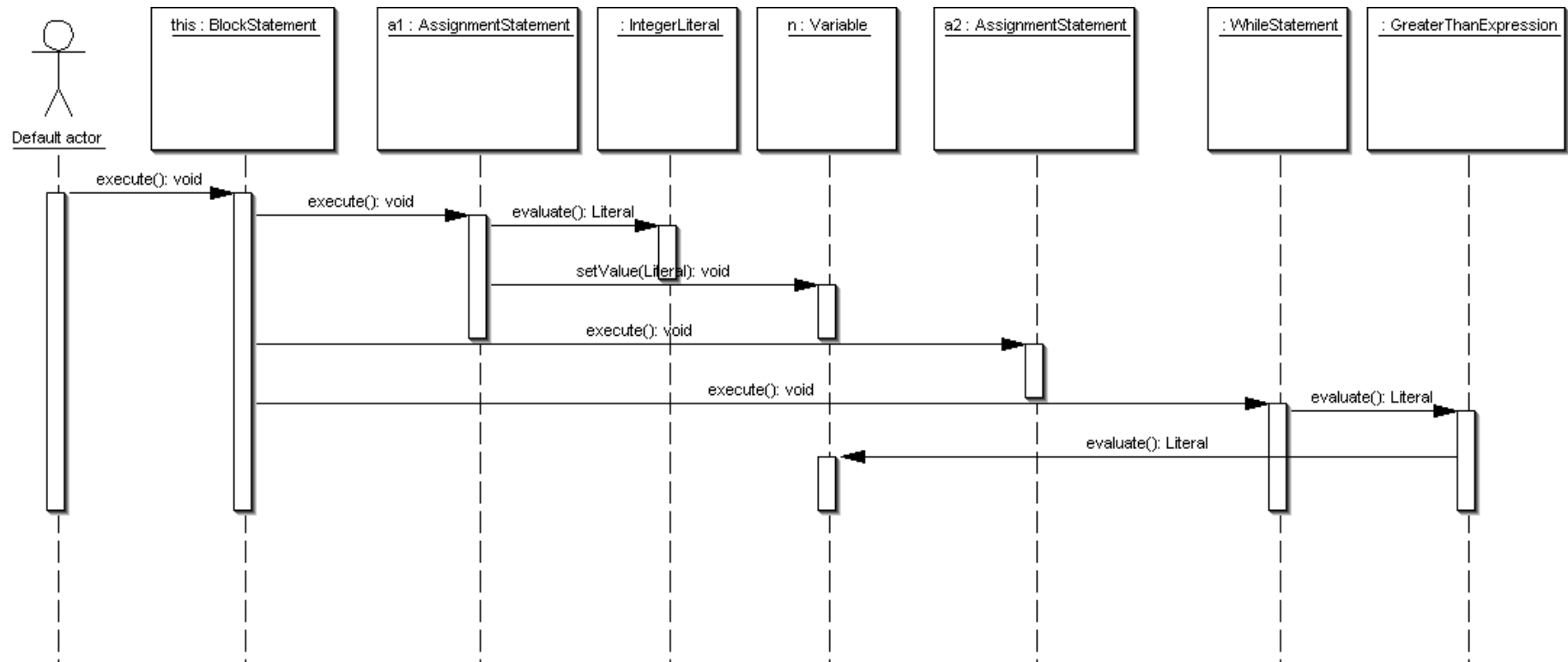


Uso dei diagrammi UML per comprendere esercizio

- **In cosa consiste un programma scritto con il progetto dato?**
 - Studiare l'esempio di calcolo del fattoriale in Test
 - Usare un diagramma degli oggetti per rappresentarlo
- **Come faccio a seguire il funzionamento dei metodi polimorfi execute e prettyPrint?**
 - Usare diagramma delle sequenze applicato al metodo execute sul blocco di istruzioni che calcola il fattoriale

Diagramma degli oggetti del programma fattoriale

Diagramma delle sequenze di execute su fattoriale



Osservazioni e domande

- **Notare che seguendo le lampadine di Eclipse si riescono a creare tutte le classi che servono per compilare il progetto.**
- **Poi, usando i suggerimenti quando scrivete “oggetto punto” si viene indirizzati bene anche nella scrittura dei metodi `execute/evaluate` e `prettyPrint`.**
- **Osservare che posso implementare come prima classe, ad esempio, `WhileStatement` compreso il metodo `execute` che usa `evaluate` su una espressione.**
 - Come fa Java a permettermi di compilare la classe `WhileStatement` se non ho ancora implementato nessuna espressione (e quindi nessuna `evaluate`)?

Refactoring del progetto in labss_il_model

- **Refactoring: rinominare progetto e spostarlo in una cartella con il nuovo nome: “labss_il_model”**
- **Refactoring: importare la libreria di interfacce (copia/incolla package dal progetto specifiche al nuovo)**
- **Refactoring: rinominare il package delle specifiche in:**
 - labss.lang.il.model
- **Refactoring: rinominare il package dell’implementazione in:**
 - labss.lang.il.model.impl

Osservazioni e domande

- **Notare che tutti i costruttori nella soluzione di riferimento hanno i parametri.**
- **Avrei potuto usare dei costruttori senza parametri e aggiungere dei metodi setter?**
 - Cosa cambia nell'implementazione
 - Cosa cambia a livello di design?

Astrazione operazioni binarie e literal

- **Osservazione: il diagramma delle classi è molto piatto**
- **Osservare la forte somiglianza tra le implementazioni dei due tipi literal e tra le implementazioni di tutte le operazioni binarie (+, -, *, /, and, or, not, <, >, =).**
- **Quante classi concrete devo/posso fare?**
 - Una per ogni costrutto linguistico (if, while, +, ...)
 - Oppure una per tutte le operazioni binarie e una per i due literal (ad esempio: MieiExpression e MieiLiteral)
- **PS. L'istruzione "if" è disponibile in due varianti: con e senza else mi conviene fare una classe o due?**

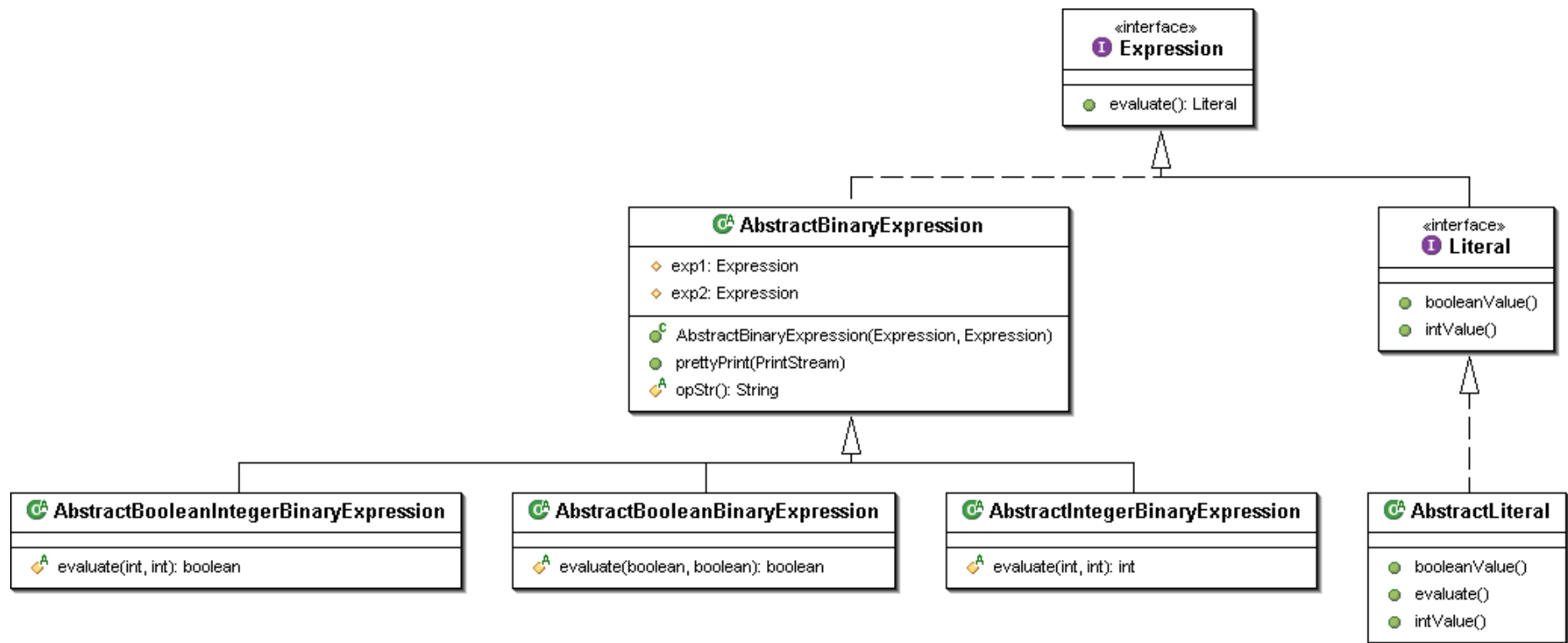
Esercizi su soluzione con classe unica

- **Come faccio a dare ai metodi `evaluate` e `prettyPrint` un comportamento polimorfo?**
- **Cosa mi conviene passare in più al costruttore: un intero e/o una stringa?**
- **E' una soluzione orientata agli oggetti?**
 - Osservazione: il metodo della `factory` sa esattamente cosa vuole costruire (ad esempio una addizione)
 - Ogni volta che eseguo `evaluate` (o `prettyPrint`) il metodo si chiede come si deve comportare (+, -, *, ...)
 - Cosa succede se aggiungo una operazione binaria?
 - Il parametro che uso per specificare l'operazione è tipato?

Soluzione che introduce classi astratte

- **Mantengo una classe per ogni costrutto linguistico ma raccolgo in una classe astratta le parti comuni alle implementazioni**
- **Refactoring: introduzione classi astratte `AbstractBinaryExpression` e `AbstractLiteral`**
- **Refactoring: pull up campi (`exp1` e `exp2`) e costruttore**
- **Osservazione: i metodi `evaluate` e `prettyPrint` pur non essendo uguali sono molto simili**
- **Posso astrarre il comportamento dei metodi?**
 - Introduzione e applicazione del design pattern `TemplateMethod`

Diagramma classi espressioni astratte



Tipi astratti vs tipi concreti

- **Notare che in tutte le classi concrete i tipi dei parametri e del risultato dei metodi, e i tipi dei campi sono tutti astratti (interfacce).**
- **Posso usare tipi concreti nei campi?**
 - Che conseguenze ho a livello di design?
- **Posso usare tipi concreti nei parametri e nei risultati?**
 - Che conseguenze ho a livello di design?
 - Posso avere diverse implementazioni del progetto compatibili tra loro?
- **Principio: program to an interface, not an implementation**
 - Posso astrarmi ovunque dalle classi concrete?

Istanziamento delle classi concrete

- **Posso istanziare solo classi concrete.**
 - Cosa vuole dire `new Expression() {...}`?
 - istanzia una interfaccia?
 - Posso nascondere al cliente le operazioni di istanziazione?
- **La factory nel progetto serve per controllare l'istanziazione delle classi concrete.**
 - Posso istanziare direttamente le classi concrete del Test?
 - Viceversa, posso obbligare i clienti ad usare la factory?
- **Refactoring: estrai metodo `fact()` da test**
- **Introduzione ai design pattern creazionali**
 - Abstract Factory, FactoryMethod

Controllo sul numero di istanze

- **Di quante istanze di Factory posso avere bisogno?**
 - Posso imporre ai clienti l'uso di una sola istanza?
- **Come si riconosce una classe che posso istanziare una volta per tutte?**
 - Posso istanziare solo due oggetti per le costanti booleane?
- **Introduzione ed uso design pattern Singleton**
- **Un Singleton si istanzia in modo diverso da una classe normale (con un metodo o campo statico)**
 - Posso rendere singleton una classe senza che i miei clienti se ne accorgano? (design pattern Monostate)

Osservazioni sul controllo della visibilità

- **I modificatori di visibilità (public, protected, *package*, private) sono applicabili ai campi, ai metodi, ai costruttori e alle classi (concrete, astratte, interfacce).**
- **Usando i modificatori di visibilità posso *imporre* le mie scelte di design. Esempi:**
 - Nascondo i costruttori delle classi gestite da factory
 - Uso private per singleton e package per factory
 - Rendo final i metodi template e protette le operazioni ridefinibili che introducono.
 - Rendo privati i campi delle classi esponendo eventualmente dei metodi per accedervi (getters/setters)
- **L'uso del modificatore “final” mi aiuta ulteriormente ad imporre le mie scelte (esempi Template Method, costanti)**

Possibili linee di sviluppo del progetto

- **Cosa devo modificare per aggiungere/modificare un costrutto al linguaggio (ad esempio l'istruzione *for*)?**
 - Posso rendere modulare l'aggiunta?
 - Introduzione al design pattern Template Manager
 - Introduzione ai design pattern Prototype e Prototype Manager
- **Cosa devo modificare per aggiungere/modificare una operazione sul linguaggio (ad esempio *typeCheck*)?**
 - Posso rendere modulare l'aggiunta?
 - Introduzione ai design pattern Visitor e Iterator
- **Posso rendere modulari entrambe le aggiunte di cui sopra?**

Supporto per Factory estendibili

- **La LanguageFactory (Abstract Factory) ora costruisce istanze di singoli costrutti**
 - Definisce un metodo per ogni costrutto
 - I metodi in genere hanno dei parametri
- **Insufficienza del pattern Abstract Factory**
 - Se voglio permettere al cliente di aggiungere degli esempi di codice istanziabili su richiesta
 - Se voglio fare un editor e supportare le operazioni di copia/incolla (e Drag and Drop) di frammenti di codice
- **Ad uso interno continuo ad usare anche la Abstract Factory**
 - Nella definizione degli esempi e dei prototipi per le factory estendibili

Factory estendibile per esempi di codice

- **Aggiungere interfaccia *TemplateFactory* con un metodo *create* per costruire un esempio di codice (template):**
 - `LanguageEntity create() // factory method`
- **Aggiungere classe *TemplateManager* con i metodi:**
 - `LanguageEntity create(String name)`
 - `void put(String name, TemplateFactory templateFactory)`
 - `void remove(String name)`
- **L'implementazione può usare una *Map* per mappare i nomi dei template sulle factory da usare per costruirli.**
- **Osservare che *create* mi restituisce un *LanguageEntity* e non mi permette di passare degli argomenti**
 - Come si può rimediare?

Osservazioni

- **Il Template Manager è più adatto per costruire esempi di codice piuttosto che singoli costrutti**
 - Un esempio in genere non richiede parametri
 - Posso accedere a tutti i vostri esempi senza concordare con voi quanti sono e come si chiamano
- **I template vengono costruiti solo se richiesti**

Factory estendibile per frammenti di codice

- **Solo a runtime conosco il frammento di codice che voglio aggiungere alla factory**
- **Ho bisogno di un meccanismo per fare una copia di un frammento di codice dato**
 - Introduzione design pattern Prototype
- **.. e di un Prototype Manager che mi permetta di aggiungere prototipi e di clonarli su richiesta**
 - Quanto è simile al Template Manager?

Meccanismo di duplicazione (clone)

- **Ho due meccanismi di creazione: istanziazione e clonazione**
 - Linguaggi class based vs. linguaggi object based
 - Anche un linguaggio class based può avere la clonazione
- **La *new* crea un oggetto (istanza) a partire da una “ricetta”: la classe.**
 - Esempio: Expression var = **new** Variable(“i”);
 - Posso passare parametri al costruttore;
 - Il risultato ha un tipo concreto
- **La *clone()* crea un oggetto facendo una copia di un oggetto già esistente.**
 - Esempio Expression var2 = (Variable) var.clone();
 - L’oggetto è già configurato bene, eventualmente lo modifico in seguito usando dei metodi (setters)
 - Il risultato ha tipo Object.

La clone di Java

- **clone() è un metodo definito in Object (tutti lo ereditano)**
 - Copia la memoria dove risiede l'oggetto.
 - Per poterlo usare bisogna implementare l'interfaccia *Cloneable* (che non definisce nessun metodo)
 - E' *protected*, per chiamarlo da fuori bisogna ridefinirlo come pubblico
 - Può provocare l'eccezione *CloneNotSupportedException*
- **Shallow vs deep clone**
 - Un oggetto può contenere riferimenti ad altri oggetti
 - Si dice profonda (deep) una clone che copia anche gli oggetti raggiungibili tramite riferimenti
 - La clone di Java è shallow, la si può ridefinire deep a piacere

Clonazione di un frammento di codice

- **Per clonare un frammento di codice ho bisogno di una clone profonda**
 - Posso rendere pubblico e usabile il metodo clone in una classe astratta ed estenderlo in tutti i costrutti del linguaggio
 - Ogni tipo concreto deve far proseguire la copia in profondità in tutti i suoi campi strutturali (di tipo riferimento)
- **Esercizio: aggiungere al progetto il supporto alla clonazione**

Manager di prototipi

- **Aggiungere classe *PrototypeManager* con i metodi:**
 - LanguageEntity create(String name)
 - void put(String name, LanguageEntity prototype)
 - void remove(String name)
- **Osservare che *create* mi restituisce un *LanguageEntity* e non mi permette di passare degli argomenti**
- **Osservazione: mi farebbe comodo definire dei prototipi incompleti. Uso dei *null*? Design pattern Null Object**
 - Aggiungere costrutti “null” NullStatement e NullExpression in modo che la prettyPrint, la clone e la execute vadano mentre le altre operazioni provochino una eccezione
- **Esercizio: provare ad integrare le funzionalità del PrototypeManager nel TemplateManager**

Osservazioni

- **Costruisco costrutti ma posso definire anche prototipi più complessi**
 - Esempio, vedi assistenza alla generazione di sorgenti di Eclipse (creazione costruttori, metodi di accesso, ...)
- **Posso aggiungere nuovi prototipi (anche a runtime)**
 - Posso introdurre nuovi costrutti aggiungendoli direttamente tra i prototipi (senza classi del modello)
 - L'utente dell'editor può chiedere che un blocco di codice venga aggiunto ai prototipi, in modo da usarlo velocemente tutte le volte che gli serve
 - I prototipi vengono costruiti e allocati anche se nessuno li richiede
- **Esercizio: la clone di Variable rende non funzionante execute(). Come posso rimediare?**

API specifiche per manipolare il modello

- **Il modello deve essere navigabile/modificabile anche dall'esterno**
 - Per supportare factory estendibili
 - per supportare operazioni modulari
- **Aggiungere a tutti i costrutti del modello i getter e i setter pubblici per tutte le proprietà**
 - BlockStatement è una sequenza di istruzioni. Che operazioni definisco per manipolarlo?
- **Osservare che le API specifiche non fanno parte delle interfacce. Chi può usarle?**
 - `((WhileStatement) lf.createWhile(..., ...)).setExp(...);`
 - `((WhileStatement) whileStm.clone()).setExp(...);`
- **Posso aggiungere tutti i metodi di manipolazione all'interfaccia LanguageEntity per rendere più usabile il modello?**
 - Il design pattern Composite suggerisce di aggiungere solo i metodi per manipolare le collezioni

Aggiunta modulare di operazioni (behavior)

- **Le varianti di ciascuna operazione polimorfa ora sono sparse su tutti i costrutti; vogliamo renderle modulari.**
- **Esercizio: riscrivere l'operazione prettyPrint in una classe separata dai costrutti (in un packagevisitors)**
 - Soluzione basata su *if* annidati di *instanceof*
- **Osservare uso dei parametri e uso delle variabili di istanza nelle due soluzioni. Cosa serve? Chi è più efficiente?**
- **Osservare accesso alle proprietà dei costrutti nelle due soluzioni. Chi è più efficiente?**
- **Osservare meccanismi di condivisione di varianti polimorfe**
- **Posso rendere costante e trascurabile il costo del dispatching?**

Soluzione basata su double dispatching (Visitor)

- **Introduzione design pattern Visitor**
- **Implementazione:**
 - Aggiungere interfaccia *LanguageVisitor* con un metodo per ogni costrutto linguistico (xxx) secondo il seguente schema:
 - void visit(XXX) oppure void visitXXX(XXX)
 - Aggiungere in LanguageEntity metodo:
 - void accept(LanguageVisitor)
 - Implementare accept in tutti i costrutti.
 - Non basta in uno astratto ereditato da tutti?
- **Scrivere *PrettyPrintVisitor* ispirandosi a *prettyPrint()*.**
 - LanguageVisitor trade-off visit con tipi concreti o cast
 - Osservare circolarità nelle dipendenze tra interfacce e implementazioni

Soluzione basata su type switch

- **Introdurre una enumerazione tipata dei costrutti che compongono il linguaggio**
 - Introduzione design pattern Enumeration
 - Implementazione con classi Enum e EnumValue
 - Aggiungere i seguenti metodi in LanguageEntity:
 - EnumValue getType()
 - int getOrdinal()
- **Usare enumerazione per sostituire gli *if* annidati di *instanceof* con uno *switch* e dei *case* sui valori della enumerazione.**

Operazioni sparse su modello vs modulari

- **Mi conviene sostituire tutte le operazioni del modello con altrettante operazioni modulari?**
- **Posso ancora avere diverse implementazioni compatibili usando i Visitors? E con il *type switch*?**
- **Possono convivere le diverse soluzioni viste?**
 - Quali metodi mi conviene definire modulari e quali lasciare sparsi sul modello?