

Design Pattern

- I design pattern sono strutture software (ovvero micro-architetture) per un piccolo numero di classi che descrivono soluzioni di successo per problemi ricorrenti
 - Tali micro-architetture specificano le diverse classi ed oggetti coinvolti e le loro interazioni
- Si mira a riusare un insieme di classi, ovvero la soluzione ad un certo problema ricorrente, che spesso è costituita da più di una classe
- Durante la progettazione, le conseguenze sulle classi di varie scelte potrebbero non essere note, e le classi potrebbero diventare difficili da riusare o non esibire alcune proprietà
- Esistono tanti cataloghi di design pattern, per vari contesti
- Sistemi centralizzati, concorrenti, distribuiti, real-time, etc.

1

Prof. Tramontana - Marzo 2020

Descrizione Di Un Pattern

- Un design pattern nomina, astrae ed identifica gli aspetti chiave di un problema di progettazione, le classi e le istanze che vi partecipano, i loro ruoli e come collaborano, ovvero la distribuzione delle responsabilità
- La descrizione include cinque parti fondamentali
- **Nome:** permette di identificare il design pattern con una parola e di lavorare con un alto livello di astrazione, indica lo scopo del pattern
- **Intento:** descrive brevemente le funzionalità e lo scopo
- **Problema** (Motivazione + Applicabilità): descrive il problema a cui il pattern è applicato e le condizioni necessarie per applicarlo
- **Soluzione:** descrive gli elementi (classi) che costituiscono il design pattern, le loro responsabilità e le loro relazioni
- **Conseguenze:** indicano risultati, compromessi, vantaggi e svantaggi nell'uso del design pattern

3

Prof. Tramontana - Marzo 2020

Design Pattern

- Un design pattern descrive un problema di progettazione ricorrente che si incontra in specifici contesti e presenta una soluzione collaudata generica ma specializzabile
- Documentano soluzioni già applicate che si sono rivelate di successo per certi problemi e che si sono evolute nel tempo
- Aiutano i principianti ad agire come se fossero esperti
- Supportano gli esperti nella progettazione su grande scala
- Evitano di re-inventare concetti e soluzioni, riducendo il costo
- Forniscono un vocabolario comune e permettono una comprensione dei principi del design
- Analizzano le loro proprietà non-funzionali: ovvero, come una funzionalità è realizzata, es. affidabilità, modificabilità, sicurezza, testabilità, riuso

2

Prof. Tramontana - Marzo 2020

Descrizione

- Nella sezione Problema della descrizione di un design pattern si parla di forze (come in fisica). Ovvero, le forze sono obiettivi e vincoli, spesso contrastanti, che si incontrano nel contesto di quel design pattern
- Altre parti della descrizione di un design pattern possono essere
- Esempi di utilizzo: illustrano dove il design pattern è stato usato
- Codice: fornisce porzioni di codice che lo implementano

4

Prof. Tramontana - Marzo 2020

Organizzazione

- I design pattern sono organizzati sul catalogo (libro GoF) in base allo scopo
- **Creazionali**: riguardano la creazione di istanze
 - Singleton, Factory Method, Abstract Factory, Builder, Prototype
- **Strutturali**: riguardano la scelta della struttura
 - Adapter, Facade, Composite, Decorator, Bridge, Flyweight, Proxy
- **Comportamentali**: riguardano la scelta dell'incapsulamento di algoritmi
 - Iterator, Template Method, Mediator, Observer, State, Strategy, Chain of Responsibility, Command, Interpreter, Memento, Visitor

5

Prof. Tramontana - Marzo 2020

Design Pattern Creazionali

- Permettono di astrarre il processo di creazione oggetti: rendono un sistema indipendente da come i suoi oggetti sono creati, composti, e rappresentati
- Sono importanti se i sistemi evolvono per dipendere più su composizioni di oggetti che su ereditarietà tra classi
 - L'enfasi va dal codificare un insieme fissato di comportamenti verso un più piccolo insieme di comportamenti fondamentali componibili
- Incapsulano conoscenza sulle classi concrete che un sistema usa
- Nascondono come le istanze delle classi sono create e composte

6

Prof. Tramontana - Marzo 2020

Factory Method

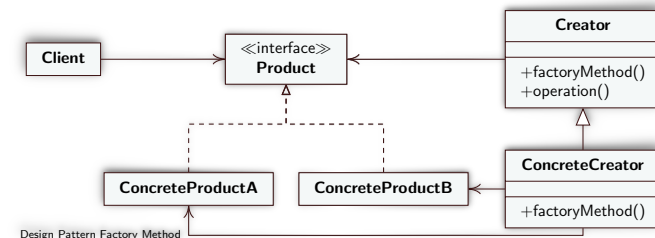
- Intento
 - Definire una interfaccia per creare un oggetto, ma lasciare che le sottoclassi decidano quale classe istanziare. Factory Method permette ad una classe di rimandare l'istanziamento alle sottoclassi
- Problema
 - Un framework usa classi astratte per definire e mantenere relazioni tra oggetti. Il framework deve creare oggetti ma conosce solo classi astratte che non può istanziare
 - Un metodo responsabile per l'istanziamento (detto **factory**, ovvero fabbricatore) incapsula la conoscenza su quale classe creare

7

Prof. Tramontana - Marzo 2020

Factory Method

- Soluzione
 - **Product** è l'interfaccia comune degli oggetti creati da factoryMethod()
 - **ConcreteProduct** è un'implementazione di Product
 - **Creator** dichiara il factoryMethod(), quest'ultimo ritorna un oggetto di tipo Product. Creator può avere un'implementazione si default del factoryMethod() che ritorna un certo ConcreteProduct
 - **ConcreteCreator** implementa il factoryMethod(), o ne fa override, sceglie quale ConcreteProduct istanziare e ritorna tale istanza

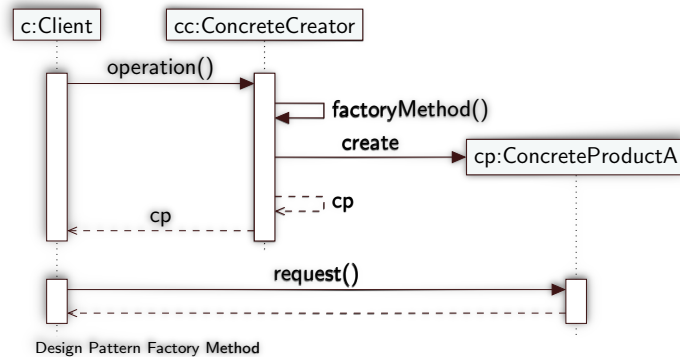


8

Prof. Tramontana - Marzo 2020

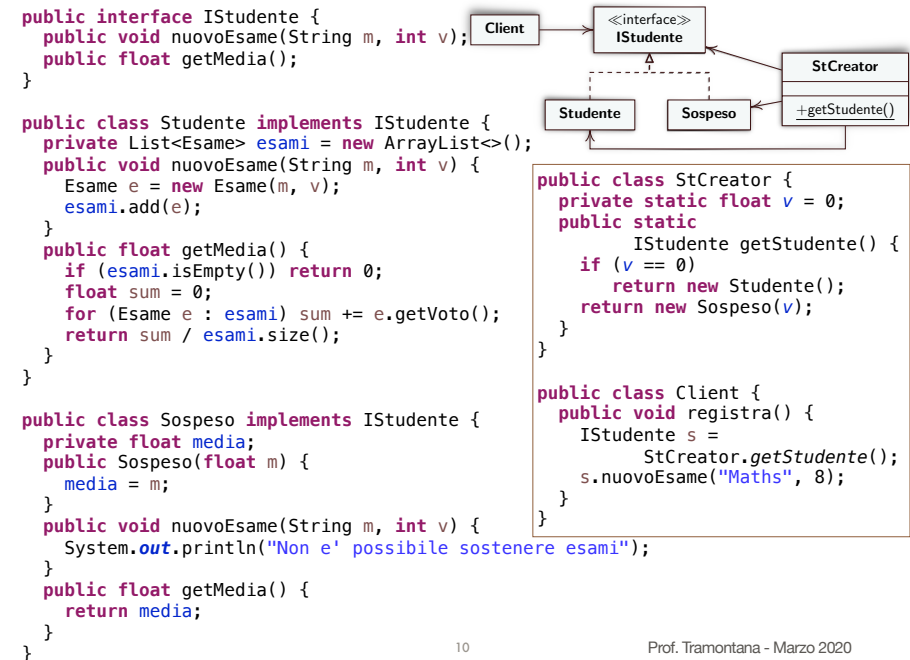
Factory Method

- Soluzione: diagramma UML di sequenza, che illustra le interazioni fra i vari ruoli



9

Prof. Tramontana - Marzo 2020

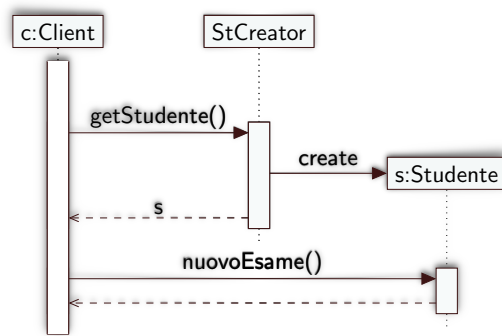


10

Prof. Tramontana - Marzo 2020

Esempio Di Factory Method

- Nel precedente esempio di codice, l'interfaccia IStuente svolge il ruolo *Product*, le classi Studente e Sospeso svolgono il ruolo *ConcreteProduct*, e la classe StCreator svolge il ruolo *ConcreteCreator*



11

Prof. Tramontana - Marzo 2020

Factory Method

- Varianti
 - Il ruolo Creator e ConcreteCreator sono svolti dalla stessa classe
 - Il factoryMethod() è un metodo static
 - Il factoryMethod() ha un parametro che permette al client di suggerire la classe da usare per creare l'istanza
 - Il factoryMethod() usa la *Riflessione Computazionale*, quindi Class.forName() e newInstance(), per eliminare le dipendenze dai ConcreteProduct, la classe istanziata sarà nota a runtime
- Conseguenze
 - Il codice delle classi dell'applicazione conosce solo l'interfaccia Product e può lavorare con qualsiasi ConcreteProduct. I ConcreteProduct sono facilmente intercambiabili
 - Se si implementa una sottoclasse di Creator per ciascun ConcreteProduct da istanziare si ha una proliferazione di classi

12

Prof. Tramontana - Marzo 2020