Singleton & Factory Method

Alessandro Midolo, Ph.D. Student
alessandro.midolo@phd.unict.it

Tutorato Ingegneria del Software

A.A. 2021/2022

Design Pattern

I **design pattern** sono strutture software per un piccolo numeri di classi che descrivono soluzioni di successo per problemi ricorrenti. Specifica l'insieme di classi ed oggetti coinvolti e le loro interazioni

- Riuso di classi
- Facilita la progettazione
- Evita di re-inventare concetti e soluzioni
- Definisce degli standard comuni tra tutti
- Aiuta i principianti nel struttura il proprio codice

Descrizione di un Design Pattern

E’ buona norma identificare ogni Design Pattern con un opportuno diagramma UML che descrive le classi che le compongono e le relazioni tra esse

- **Nome:** rappresenta lo scopo del pattern
- **Intento:** funzionalità e scopo
- **Problema:** il problema a cui il pattern è applicato e le sue condizioni
- **Soluzione:** descrizione delle classi che lo compongono, le loro responsabilità e le loro relazioni
- **Conseguenze:** vantaggi e svantaggi nell’uso del design pattern

Due descrizioni aggiuntive:

- **Esempi di utilizzo:** esempi pratici di come è stato applicato il DP
- **Codice:** porzioni di codice che lo implementano

Singleton

Intento

- Assicurare che una classe abbia una sola istanza e fornire un punto di accesso globale all'istanza

Problema

- Necessità di classi che semanticamente devono rappresentare una singola istanza in tutta l'applicazione
- Una variabile globale non proibisce di avere più oggetti per una classe
- La classe deve essere responsabile di tenere traccia della sua unica istanza

Soluzione

- La classe Singleton presenta un metodo *getInstance()*
- La classe Singleton è responsabile della creazione dell'istanza
- Il costruttore della classe deve essere privato

Factory Method

Intento

- Definire un'interfaccia per creare un oggetto ma lasciare che siano le sottoclassi a decidere quale classe istanziare

Problema

- Si usano classi astratte per definire le relazioni tra oggetti differenti, ma il client conosce solo queste classi astratte e non può instanziarle
- Il metodo "*factory*" decide lui quale classe creare in base alle proprietà richieste

Soluzione

- **Product** è l'interfaccia comune degli oggetti creati dal factoryMethod()
- **ConcreteProduct** è un'implementazione di Product
- **Creator** dichiara il factoryMethod() e ritorna un oggetto di tipo Product
- **ConcreteCreator** implementa o fa l'override del factoryMethod() scegliendo quale ConcreteProduct istanziare

Diagramma UML Factory Method