Entwurfsmuster

Entwurfsmuster

- Entwurfmuster (software design patterns) sind Lösungs-Vorlagen für wiederkehrende Entwurfsprobleme
- erstmals als Konzept vorgestellt von C. Alexander (1977)
- populär insbesondere durch “Gang of Four”:
  E. Gamma, R. Helm, R. Johnson, J. Vlissides.
  Design Patterns: Elements of Reusable Object-Oriented Software (1994)
- Einteilung in drei Typen
  - Erzeugungsmuster (creational patterns): Muster zur gekapselten Objekterzeugung
  - Strukturmuster (structural patterns): Muster für Beziehungen zwischen Klassen und Objekten
  - Verhaltensmuster (behavioral patterns): Muster für Kommunikation zwischen Objekten
- es wurden auch weitere Typen und weitere Einteilungs-Logiken vorgeschlagen
  → Software-Technik
- hier nur Auswahl von Entwurfsmustern, die für Projekte vermutlich nützlich sind

Erzeugungsmuster – Factory-Methode

Situation

- Nutzung eines Objektes mit bestimmten Eigenschaften in *Creator*-Klasse, z.B. definiert über Interface oder abstrakte Klasse
- Konkrete Implementierung soll in Unterklassen von *Creator* festgelegt werden können

Umsetzung

- Interface oder abstrakte Klasse für “Art” der Objekte
- In Creator-Klasse (oder -Interface) Definition einer Methode zur *Erzeugung* des Objektes→ **Factory-Methode**
- Aufruf der Factory-Methode an Stellen im Code, an denen Objekt(e) benötigt werden
- In Unterklassen der Creator-Klasse muss die Factory-Methode implementiert bzw. kann sie überladen werden
- Abwandlung: Factory-*Klasse* mit (static) Methode(n) zur Objekt-Erzeugung

Erweiterung: Abstrakte Factory

- lagere Factory-Methode(n) in eigene abstrakte Klasse aus
- übergib Instanz einer konkreten Implementierung dieser abstrakten Klasse

Erzeugungsmuster – Singleton

Situation

- Klasse soll nur eine Instanz haben, z.B. um Speicher zu sparen oder Konsistenz sicherzustellen
- Instanz soll leicht zugreifbar sein

Umsetzung

- alle Konstruktoren `private`
- Instanz üblicherweise als `private static` Variable
- `public static`-Methode zum Zugriff auf die (eine) Instanz
- ggf. mit lazy initialization

Erweiterung

- Multiton: mehrere, aber begrenzte Anzahl von, Instanzen einer Klasse

Beispiel: `Singleton` und `SingletonLazy`

Strukturmuster – Adapter

Situation

- Wir wollen eine Klasse/Implementierung A verwenden, aber benötigen dafür ein bestimmtes Interface B,
  - das diese Klasse nicht implementiert,
  - dessen Funktionalität aber prinzipiell in der Klasse existiert

Umsetzung

- Objekt-Adapter: Adapter-Klasse
  - implementiert benötigtes Interface B
  - enthält Instanz der vorhandenen Klasse A und setzt Aufrufe der Methoden aus B über Aufrufe von Methoden aus A auf der enthaltenen Instanz um
- Klassen-Adapter: Adapter-Klasse
  - implementiert benötigtes Interface B und erweitert existierende Klasse A
  - leitet Aufrufe der Methoden aus B an bereits implementierte Methoden aus A weiter

Beispiel: `StringTokenizerIterator`

Strukturmuster – Kompositum

Situation

- es sollen sowohl Einzel-Objekte als auch zusammengesetzte Objekte existieren
- beide sollen (auf einer hohen Abstraktionsebene) auf die gleiche Art angesprochen werden können

Umsetzung

- Definition eines allgemeinen/gemeinsamen Interfaces für Einzel- und zusammengesetzte Objekte
- Einzelobjekte implementieren dessen Methoden direkt
- zusammengesetzte Objekte greifen dabei auf enthaltene Einzel-Objekte zurück

Beispiel: `SequenceRegion`, `Exon` (Einzel-Objekt), `Transcript` (zusammengesetztes Objekt)

Verhaltensmuster – Iterator

Situation

- es soll über die Elemente eines Objektes (Liste, Menge, Graph, ...) iteriert werden
- Implementierung soll nicht abhängig von konkreten Datenstrukturen sein

Umsetzung

- Objekt hat Methode zur Generierung (und Rückgabe) des Iterators
- Iterator kennt interne Datenstruktur und stellt Methoden zum iterieren bereit

Beispiel: `Iterable<T>` Interface in Java:
`https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Iterable.html`

Verhaltensmuster – Schablonenmethode

Situation

- ein Algorithmus soll generisch implementiert werden
- Details der Implementierung sollen variabel in Unterklasse(n) festgelegt werden

Umsetzung

- Implementierung des Algorithmus in (abstrakter) Oberklasse
- Definition von abstrakten Methoden für Details
- Unterklassen füllen die Details in ihrer Implementierung

Beispiel: `public int getNext(int node, Set<Integer> reached)` in `Graph` Interface aus Seminar 5

Verhaltensmuster – Strategie

Situation

- zur Lösung eines Problems existieren mehrere Strategien, z.B. verschiedene Sortieralgorithmen
- für eine konkrete Probleminstanz soll eine dieser Strategien zur Laufzeit gewählt werden

Umsetzung

- Definition eines Interfaces für alle Strategien
- Implementierung des Interfaces für verschiedene Strategien
- in Abhängigkeit von der Eingabe wird im Code eine Strategie ausgewählt
- an anderen Stellen im Code werden nur Methoden des Interfaces verwendet

Beispiel: `IntersectStrategy` mit `IntersectSets` und Implementierungen `IntersectSmall` und `IntersectLargeComparable`

Verhaltensmuster – Beobachter

Situation

- der Zustand eines Objektes A kann sich flexibel ändern
- vom Zustand dieses Objektes hängen weitere Objekte X_1,...,X_N ab
- bei Zustandsänderung sollen alle abhängigen Objekte informiert werden

Umsetzung

- abhängige Objekte implementieren ein allgemeines Beobachter-Interface
- abhängige Objekte X_1,...,X_N registrieren sich bei Objekt A
- wenn sich der Zustand von A ändert, informiert A die abhängigen Objekte über Methode des Beobachter-Interfaces
- Varianten
  - Objekte holen sich den neuen Zustand über Methoden von A ab
  - Methode des Beobachter-Interfaces erlaubt Kommunikation des neuen Zustands
- häufig für grafische Oberflächen etc. verwendet