Software-Produktlinien

Übung 2: Laufzeitvariabilität und Entwurfsmuster

Elias Kuiter, Gunter Saake Wintersemester 2025/26

1. Laufzeitvariabilität konfigurieren

(a) Was ist Laufzeitvariabilität?

(b) Nenne Beispiele für zwei reale Softwaresysteme, die Laufzeitvariabilität nutzen. Beschreibe, wie sie diese nutzen und wie der Benutzer sie konfiguriert.

(c) Stell dir vor, du entwickelst eine Musik-Player-App mit folgenden konfigurierbaren Optionen: visueller Equalizer (aktiviert oder deaktiviert), Streaming-Qualität (hoch oder niedrig) und Theme (hell oder dunkel).

i. Beschreibe zwei mögliche Methoden, um diese Optionen zur Laufzeit zu konfigurieren.

ii. Erkläre an einem geeigneten Beispiel, was passieren könnte, wenn widersprüchliche Optionen ausgewählt werden. Beschreibe, wie du solche Konflikte zur Laufzeit erkennen und behandeln würdest.

(d) Welche Vor- und Nachteile hat die Umsetzung von Varianten mit Laufzeitvariabilität?

2. Laufzeitvariabilität realisieren

Du entwickelst eine moderne Navigations-App, die von Tausenden von Pendlern, Touristen und Fitness-Enthusiasten täglich genutzt wird. Die App muss drei primäre Betriebsmodi unterstützen: Fahrmodus, Gehmodus und Fahrradmodus. Jeder Modus bietet eine unterschiedliche Benutzererfahrung, Routenberechnung und Schnittstellenanpassung, die auf die aktuelle Aktivität des Benutzers zugeschnitten ist. Benutzer können je nach Umgebung und Präferenzen zwischen diesen Modi wechseln, um die genauesten und relevantesten Navigationsanweisungen zu erhalten.

(a) Beschreibe, wie du diese Variabilität mit globalen Parametern umsetzen würdest.

(b) Beschreibe eine alternative Lösung mit Methodenparametern anstelle von globalen Parametern.

(c) Vergleiche die beiden Lösungen. Welche ist einfacher zu warten und welche ist flexibler für zukünftige Updates? Begründe deine Antwort.

(d) Implementiere basierend auf deinem bevorzugten Ansatz die Navigationslogik unter Verwendung des unten stehenden Gerüsts. Erkläre, welchen Variabilitätsansatz du verwendet hast (globale Konfiguration, Methodenparameter oder Design Patterns) und warum er in diesem Fall geeignet ist.

Gerüst zur Implementierung von Navigationsvariabilität

```
// Enum for travel modes
enum Mode {
  DRIVING, WALKING, CYCLING
}

class Navigator {
  // Declare configuration or strategy fields if needed

  Navigator() {
    // Initialize configuration if necessary
  }

  void navigate() { // Define the correct parameters for navigate()
    // Insert logic in conditional statements
    if () {
      System.out.println("Calculating driving route...");
    } else if () {
      System.out.println("Calculating walking route...");
    } else if () {
      System.out.println("Calculating cycling route...");
    }
  }
}

class Route {
  // Simple placeholder class for route data
}

public class Main {
  public static void main(String[] args) {
    Route route = new Route();
    Navigator nav = new Navigator();
    // Call nav.navigate() with proper configuration or arguments
    nav.navigate();

  }
}
```

3. **Design Patterns für Laufzeitvariabilität**

Du entwirfst anpassbare Software für eine Kaffeemaschine. Die Maschine kann verschiedene Getränke wie Espresso, Cappuccino und Latte zubereiten. Die Maschine kann optional Zucker oder Milch hinzufügen und Zubereitungszeiten in einer Datei protokollieren.

(a) Erkläre *Design Patterns* und deren allgemeinen Zweck! Nenne und erkläre kurz ein Design Pattern, das zur Implementierung von Variabilität verwendet werden kann.

(b) Wähle ein Design Pattern aus, das du verwenden würdest, um die Variabilität im

Kaffeemaschinen-System zu implementieren. Begründe, warum dieses Design Pattern für dieses Szenario geeignet ist.

(c) Schreibe einen kleinen Pseudo-Code-Ausschnitt oder erstelle ein Klassendiagramm, das zeigt, wie das ausgewählte Design Pattern im Kaffeemaschinen-System funktionieren würde.

(d) Stell dir vor, der Benutzer ändert das Kaffee-Rezept, während die Maschine das Getränk bereits zubereitet.

i. Erkläre, was schief gehen könnte, wenn Konfigurationen während der Zubereitung zur Laufzeit geändert werden.

ii. Schlage eine Lösung vor, um potenzielle Fehler zu verhindern, und erkläre diese.

(e) Könntest du dieses Kaffeemaschinen-System ohne Verwendung eines Design Patterns implementieren? Wenn ja, wie? Wenn nicht, warum nicht?

4. **Parameter oder Design Patterns?**

Das folgende Java-Quelltextfragment implementiert Variabilität zur Laufzeit mit globalen Konfigurationsparametern, um verschiedene Audio-Ausgaben in einem Videospiel zu ermöglichen.

Eine Methode mit konfigurierbarer Audio-Ausgabe in Java

```
void playSound(String sound) {
  if (Config.PLAY_ON_SPEAKER) {
    speakerSystem.play(sound);
  } else if (Config.PLAY_ON_HEADPHONES) {
    headphoneSystem.play(sound);
  } else if (Config.PLAY_ON_STREAM) {
    onlineStreamSystem.play(sound);
  }
}
```

Die aktuelle Implementierung muss verbessert werden, um die folgenden Anforderungen zu erfüllen:

- Das System sollte das dynamische Umschalten von Audio-Ausgaben während des Spielens ermöglichen, ohne das Spiel neu zu starten.
- Das System sollte das gleichzeitige Abspielen von Tönen auf mehreren Ausgaben unterstützen.
- Das System sollte das Hinzufügen neuer Audio-Ausgabe-Typen wie VR-Headsets ermöglichen, ohne den vorhandenen Quellcode zu ändern.

(a) Welche Probleme hat die aktuelle Implementierung? Liste die Probleme auf und erkläre sie kurz.

(b) Welches Design Pattern könnte die Probleme lösen und warum? Erkläre, wie das ausgewählte Design Pattern das Umschalten von Audio-Ausgaben zur Laufzeit, mehrere gleichzeitige Ausgaben und das Hinzufügen neuer Audio-Systeme ohne Änderung des bestehenden Codes unterstützen würde.

(c) Refaktoriere (= überarbeite) den bereitgestellten Java-Code unter Verwendung des vorgeschlagenen Design Patterns, so dass alle drei Anforderungen erfüllt werden.