Übungsblatt 8

Aufgabe 1

1. Naja halt hamster Runnerhamster und dann die 3 unterschiedlichen
2. Das Strategy Design Pattern ist ein Konzept in der Softwareentwicklung, das es ermöglicht, das Verhalten von Objekten dynamisch zu ändern. Anstatt verschiedene Verhaltensweisen durch Unterklassen zu repräsentieren, werden diese in separaten Strategieklassen gekapselt. Diese implementieren ein gemeinsames Interface, was es erlaubt, Strategien zur Laufzeit auszutauschen.

Das Strategy Design Pattern bietet gegenüber dem Einsatz von Unterklassen mehr Flexibilität und Erweiterbarkeit. Im Gegensatz zur Vererbung ermöglicht diese Strategie dynamische Änderung des Verhaltens zur Laufzeit, ohne den Hauptcode verändern zu müssen.

Außerdem ist es so deutlich einfacher neue Verhaltensweisen zu implementieren, die man nicht immer eine neue Klasse anlegen muss, wodurch der Code „sauber“ bzw. ordentlicher gehalten wird.

(e) Jeder spezifische runner hamster hätte dann eine eigene klasse mit den strategien

Aufgabe 2

1. Defensives Programmieren bezeichnet die Herangehensweise an die Entwicklung von Computersystemen, bei der das System möglichst viele Voraussetzungen eigenständig überprüft, bevor der eigentliche Zweck erfüllt wird. Wir programmieren defensiv bei Operationen, die mit dem Nutzer „interagieren“, um dort ungewollte Fehler und oder Zustände zu vermeiden. Wichtige Schlüsselwörter sind hier public und protected, da sie uns anzeigen, dass wir hier defensiv programmieren sollten, sowie if-else, try-catch und z.B. die IllegalArgumentException beim Implementieren.

Offensives Programmieren hingegen setzt präventive Maßnahmen ein, um Fehler zu verhindern, bevor sie auftreten. Es beinhaltet das proaktive Identifizieren potenzieller Probleme, das Validieren von Eingaben und das Implementieren von Schutzmechanismen, um unerwartete Situationen von vornherein zu adressieren. Wichtige Schlüsselwörter sind hier private und default als Sichtbarkeiten sowie assert zur Vermeidung von Fehlern (oder ungewollten Zuständen).

(f)

Mit Bezug auf das „Uniform Access Principle“ ist es egal wie die Anzahl der Türen berechnet wird. Es ist nur wichtig, dass immer die richtige Zahl ausgegeben wird. Eine Möglichkeit wäre es bei der Implementierung einer Tür einen Counter, der die Anzahl an Türen zählt zu erhöhen. Vorteil dieser Variante ist, dass die Anzahl nicht zur Laufzeit immer neu berechnet werden muss und sofort rausgegeben werden kann. Nachteil dieser Variante ist, dass extra Speicher zum Speichern dieser Zahl verwendet wird (weniger Speicher für mehr Performance). Eine andere Variante wäre es die zahl der Türen zur Laufzeit berechnen zu lassen. Vorteil dieser Variante ist, dass kein zusätzlicher Speicher verbraucht wird. Nachteil ist dass es zur Laufzeit zur einbüßen bei der Performance kommt um die zahl der Türen zu berechnen (weniger Performance für mehr Speicher).

Die erste Variante sollte man wählen, wenn die Anzahl der Türen besonders wichtig ist und häufig ausgegeben werden muss, damit es nicht häufig zu Performance issues kommt. Wenn die Anzahl der Türen allerdings recht irrelevant ist und selten ausgegeben wird, kann man den Speicher sparen und zu seltenen Anlässen berechnen lassen.