Semesterprojekt 2020 - Spaceshooter

Als Semesterprojekt wurde mit Processing ein kleines Spiel entwickelt. Der Spieler kann Raumschiffe anklicken, welche dann den Punktestand erhöhen. Es wurde jackson als zusätzliche Bibliothek verwendet.

Klassenhierarchie

Es gibt mehrere Vorkommen von Klassenhierarchien mit 3 Ebenen. Zum einen ist die Klasse CollisionHandler von InitHandler und ProcessingPainter von CollisionHandler abgeleitet.
Diese Hierarchie wurde gewählt, da CollisionHandler und ProcessingPainter Attribute benötigen, die auch InitHandler benötigt, zudem verwendet ProcessingPainter Funktionalitäten von InitHandler und CollisionHandler. Hätte man nur eine Klasse verwendet, wäre diese recht umfangreich und unübersichtlich geworden. Diese drei Klassen befinden sich in einem gemeinsamen Package. CollisionHandler und InitHandler sind beide Package Private, so dass diese Klassen nicht instanziiert werden können. Sie können nur über ProcessingPainter genutzt werden.

Es gibt zwei weitere Klassenhierarchien mit 3 Ebenen. Visible sowie Drawable SpaceObjects stellen die Super-Klassen da. Von Ihr sind Visible / Drawable SpaceShip abgeleitet. Von SpaceShip sind wiederum UFO und BattleShip abgeleitet. Zudem sind von SpaceObjects zusätzlich die Klasse Asteroid abgeleitet. Dies ermöglicht es nun zwei Listen zu verwenden, welche DrawableSpaceObjects und VisibleSpaceObjects enthalten. Unter Verwendung von einem Visitor Design Pattern und Polymorphie kann zur Laufzeit ermittelt werden, ob es sich um ein Asteroid, Ufo oder Battleship handelt.

Polymorphie

Polymorphie wurde hauptsächlich bei der Erstellung der Listen, welche Drawable / Visible SpaceObjects enthalten, verwendet. Jedoch wurde Polymorphie auch genutzt, um nach Feststellung eines speziellen SpaceOjects, durch downcasting Methoden aufzurufen, die erst in abgeleiteten Klassen implementiert wurden.

Design Pattern

Es wurden MVC, Singleton, Factory, Visitor, sowie vereinzelt das Dependency Injection Design Pattern verwendet.

Singleton
Das Design Pattern Singleton wurde dann genutzt, wenn von vornherein klar war, dass nur eine einzige Instanz einer Klasse gebraucht wird. Von der Klasse Json, die in der Lage ist InputStream Objekte in JsonNode Objekte und JsonNode Objekte in Java Objekte zu konvertieren, wird im gesamten Projekt nur eine Instanz benötigt. Genauso verhält es sich z.B. bei der Klasse SpaceShipFactory.

Factory
Das Factory Design Pattern wurde für die Erstellung von Drawable / Visible SpaceObject in der Klasse InitHandler verwendet. Durch Nutzung dieses Design Patterns kann klar definiert werden, welches

SpaceObject erstellt werden soll und es lässt sich beliebig erweitern. Zudem könnten andere SpaceShips zurückgegeben werden ohne den Rest des Codes zu sehr verändern zu müssen.

Visitor
Das Visitor Design Pattern wurde verwendet, um zur Laufzeit zwischen den einzelnen SpaceObjects unterscheiden zu können. Der Rückgabewert ist ein Integer, der je nach Art des speziellen SpaceObjects variiert. So ist es möglich, abhängig von der Art des SpaceShips den Punktestand anzupassen. Hierzu hätte auch instanceOf verwendet werden können, jedoch wurde uns in der Vorlesung von der Nutzung abgeraten, da die Nutzung sehr “teuer“ sein soll.

Dependency Injection
Dieses Design Pattern wurde z.B. für die Methode parse(InputStream) aus der Json Klasse verwendet, um alle Arten von InputStreams verwenden zu können. Für die Painter Klasse, die Teil des Controllers ist, wurden Arraylists, ein Array und URLs “injiziert“. Die Nutzung dieses Design Patterns bietet Flexibilität sowie Vorteile beim Testen.

Darstellungs-Logik / Business-Logik

Drawable Objekte verwenden Methoden, welche die Darstellung bzw. die Position der einzelnen Objekte verändern. Zudem nutzen sie Attribute, die für die Darstellung wichtig sind wie z.B. Position und Geschwindigkeit. Die Visible Objekte verwenden Attribute für die Bestimmung der Höhe und Breite.