Programmieren 2

– Internationaler Studiengang Medieninformatik –

1. Laborübung: Anwendungskern

Ausgabe: 18.03.2013

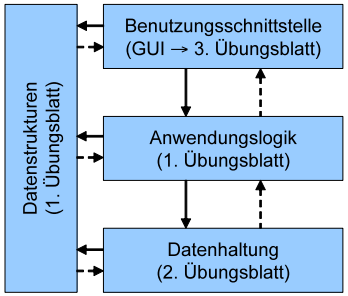
Allgemeines:

Die Laborübungen zur Veranstaltung „Programmieren 2“ werden in Form eines Projekts durchgeführt, das in der Regel jeweils 3 Studierende gemeinsam bearbeiten (siehe auch Foliensatz *PROG_2_00 – Organisatorisches*). Sie haben grundsätzlich die Wahl zwischen zwei Projekten:

- *eShop*: Ein Shop-System, über das Kunden vorhandene Artikel kaufen können. Der Artikelbestand kann dabei durch Mitarbeiter der Anbieterfirma verwaltet werden.
- *Risiko*: Das bekannte Brettspiel als Computerspiel

Nach individueller Absprache besteht in Ausnahmefällen auch die Möglichkeit, ein von Ihnen gewähltes Projekt zu bearbeiten (z.B. Trivial Pursuit, Monopoly). Aber Vorsicht, unterschätzen Sie die Komplexität derartiger Projekte nicht!

Zu beiden genannten Projekten wird es insgesamt vier Übungsblätter geben, die abhängig vom Fortschreiten der Vorlesung ausgegeben werden und der Steuerung Ihrer Entwicklungsaktivitäten dienen sollen. Die ersten drei Übungsblätter orientieren sich an der 3-Schichten-Architektur, die in der folgenden Abbildung durch eine vierte vertikale „Schicht“ für die auf allen übrigen Schichten erforderlichen Datenstrukturen ergänzt wird:



Ein viertes Übungsblatt ergänzt schließlich noch den Datenaustausch über ein Netzwerk.

Wichtig für alle Aufgaben: Gehen Sie sicher, dass Sie den jeweiligen Problembereich genau verstanden haben. Modellieren Sie zu diesem Zweck zunächst das Problem (rudimentär in UML-Notation wie in der VL), bevor Sie in die Tasten hauen!!

Hochschule Bremen
Int. Studiengang Medieninformatik
Sommersemester 2013
Dozent: Thorsten Teschke

Projekt „eShop“:

Aufgabe 1: Anwendungskern + CUI

Im ersten Schritt soll zunächst der Anwendungskern entwickelt werden, in dem die für den eShop relevanten Datenstrukturen wie z.B. Artikeldaten (vertikale „Schicht“) und die darauf arbeitenden Operationen wie z.B. das Ein- und Auslagern von Artikeln (Anwendungslogik, 2. Schicht) sinnvoll in Klassen zusammengefasst werden. Beachten Sie, dass diese Klassen ggf. miteinander kooperieren müssen, indem sie gegenseitig Methoden aufrufen.

Die (minimalen) fachlichen Anforderungen an den eShop in der Übersicht:

- In einem eShop werden *Artikel* mit Bezeichnung, eindeutiger Artikelnummer und ihrem Bestand gespeichert.
- Die Artikel können nach Bezeichnung oder Artikelnummer *sortiert* auf dem Bildschirm *ausgegeben* werden (Ausgabe → Aufgabe der Benutzungschnittstelle auf Schicht 1).
- *Mitarbeiter* (mit Name und eindeutiger Nummer) können *neue Artikel anlegen* und den *Bestand existierender Artikel erhöhen*.
- *Kunden* (ebenfalls mit Name und Nummer, aber auch mit Adresse) können (mehrere!) *Artikel in ihren Warenkorb legen*, die *Stückzahl* von Artikeln im Warenkorb *ändern* und den *Warenkorb leeren*.
  Hinweis: Mal über sinnvolle Generalisierung nachdenken, wenn zwei oder mehr Klassen gleiche Attribute oder Methoden aufweisen.
- Kunden können im Warenkorb enthaltene *Artikel kaufen*, wobei der Warenkorb geleert wird und die Artikel aus dem Bestand genommen werden. Es wird ein *Rechnungsobjekt* erzeugt und auf dem Bildschirm *ausgegeben*. Das Rechnungsobjekt enthält u.a. Kunde, Datum, die gekauften Artikel inkl. Stückzahl und Preisinformation sowie den Gesamtpreis.
- Jede *Ein- und Auslagerung wird als Ereignis* mit Datum (Nummer des Jahrestags reicht), Anzahl, betroffenem Artikel und sowie beteiligtem Mitarbeiter bzw. Kunden festgehalten.

Damit der Anwendungskern getestet werden kann, soll zudem eine CUI („Command Line User Interface“) entwickelt werden, d.h. eine Möglichkeit zur Bedienung des eShops über die Kommandozeile.

Auf dem nächsten Übungsblatt werden diese Anforderungen – neben der dann gewünschten persistenten (d.h. dauerhaften) Datenhaltung in einer Datei – noch geringfügig ergänzt.

Projekt „Risiko“:

Aufgabe 1: Anwendungskern + CUI

Ziel des Mehrpersonenspiels *Risiko* ist es, möglichst viele Länder der Spielwelt mit mindestens einer eigenen Einheit zu besetzen bzw. eine bestimmte Aufgabe („Mission“) zu erfüllen (z.B. „Besetzen Sie alle Länder eines Kontinents“).

Im ersten Schritt soll zunächst der Anwendungskern entwickelt werden, indem die für Risiko relevanten Datenstrukturen wie z.B. Länder und Einheiten (vertikale „Schicht“) und die darauf arbeitenden Operationen wie z.B. z.B. angreifen oder verschieben (Anwendungslogik, 2. Schicht) sinnvoll in Klassen zusammengefasst werden.

Die grundlegende Spielidee (für Details bitte mal im eigenen Spieleschrank, in der Risiko-Anleitung in AULIS oder hier nachschlagen http://www.hasbro.de/manuals/14538.pdf):

- Eine *„Welt“* stellt sich als zusammenhängendes Gebilde von aneinander grenzenden *Ländern* dar, d.h. sie ist im Grunde ein Graph von Nachbarschaftsbeziehungen. Jedes Land hat einen Namen, ein Kürzel und eine *Armee (Menge von Einheiten)*, die genau einem *Spieler* gehört. Außerdem ist jedes Land einem *Kontinent* zugeordnet.
- Jeder *Spieler* besitzt eine Identität und kennt seine *Mission* (z.B. alle Länder eines Kontinents zu erobern).
- Der Spielzyklus ist wie folgt (jeweils für einen Zug eines Spielers):
  - Neue *Einheiten verteilen*: Der Spieler erhält zu Beginn jeder Runde in Abhängigkeit von den von ihm besetzten Ländern und Kontinenten neue Einheiten, die er verteilen muss.
  - *Angriff*: Eingabe, mit welcher Teilarmee (wie viele Einheiten) von welchem eigenen Land aus welches andere Land *angegriffen* wird. Ein Spieler kann mit maximal drei Einheiten zurzeit angreifen. Mindestens eine Einheit muss im eigenen Land verbleiben und darf nicht für den Angriff genutzt werden.
  - *Verteidigung*: Eingabe, mit welcher Teilarmee (wie vielen Einheiten) das angegriffene Land *verteidigt* wird. Ein Spieler kann sein Land mit maximal zwei Einheiten zurzeit verteidigen.
  - *Auswertung von Kämpfen*: Jede Einheit der Angriffsarmee „würfelt“ gegen den Wurf einer zufälligen Verteidigungseinheit. Bei echt größerer Zahl ist die Verteidigungseinheit vernichtet, sonst die Angriffseinheit.

    Der Angreifer kann seine Angriffe bis zur endgültigen Eroberung des Landes fortsetzen oder abbrechen.
  - *Einrücken*: Wenn die Verteidigungsarmee vernichtet ist, muss mindestens eine Angriffseinheit in das eroberte Land einrücken. Wichtig ist, dass mindestens eine Einheit im Land des Angreifers verbleibt.

    Der Angreifer kann beliebige andere Länder angreifen oder aber seine Eroberungsaktionen für diese Spielrunde beenden.
  - *Verschieben von Einheiten*: Eingabe, wie viele *bisher unbeteiligte(!) Einheiten* von welchem eigenen Land in welches eigene Nachbarland *verschoben* werden sollen. Dabei muss immer mind. eine Einheit in dem Land zurückbleiben. Eine Einheit darf nur einmal pro Runde verschoben werden.
  - Darüber hinaus können Informationen über die Länder abgefragt werden (Besitzer, Einheiten auf Land, vielleicht auch benachbarte Länder, ...).

Bei diesem ersten Entwicklungsschritt können Sie die Missionen, die die einzelnen Spieler erfüllen müssen, zunächst außer Acht lassen.

Damit der Anwendungskern getestet werden kann, soll zudem eine CUI („Command Line User Interface“) entwickelt werden, d.h. eine Möglichkeit zum Spielen von Risiko über die Kommandozeile. Das wird zugegebenermaßen kein schönes Spielen, ist aber zum Testen des Anwendungskerns unerlässlich.

Hochschule Bremen
Int. Studiengang Medieninformatik
Sommersemester 2013
Dozent: Thorsten Teschke

Bewertung

In die Bewertung gehen die folgenden Kriterien ein:

- Arbeit in den Laborübungen, Verständnis der Projekt(zwischen)ergebnisse (Einzelgespräch zu allen Inhalten – muss bestanden werden!)
- Komplexität der Aufgabe: für ein „sehr gut“ gelten bei einem eShop höhere Anforderungen an die „Perfektion“ und Vollständigkeit als bei Risiko
- Qualität der Software:
  - Architektur / Modellierung (Wie ist die Software strukturiert?)
    → korrekte und sinnvolle Verwendung von Klassen und Beziehungen zwischen Klassen
  - Funktionalität (Werden die gestellten Anforderungen erfüllt?)
  - Fehlerfreiheit (auch: Einsatz des Exception-Konzepts von Java)
  - Verständlichkeit des Codes (Strukturierung, Kommentare)
  - Bedienbarkeit und optische „Schönheit“ der Benutzungsoberfläche (Letzteres ist nicht das wichtigste Kriterium!)
  - Besondere Lösungsideen
- Dokumentation der Software (JavaDoc) und der Architektur (z.B. in Form von (vereinfachten) Klassen- oder Kollaborationsdiagrammen wie in der Vorlesung)
- Abschließendes Kolloquium