Inhalt

1. Soll/Ist Beschreibung

- Das System hat die Klassen Address, DataStore,Main, PhoneNumber und Student ✓
- Das System hat einen 2. Ordner namens test, welcher die Testklassen beinhaltet ✓
- Die Main-Klasse steuert die Anwendung mit Konsoleneingabemöglichkeiten ✓
- Die Studenten-Klasse ist das Zentrum der Anwendung und implementiert die Funktionalitäten, die aus der Main-Klasse aufgerufen werden ✓
- Man kann nach Studenten by ID suchen (und damit auswählen) ✓
- Man kann sich die Infos (ID, Name und Vorname) des gewählten Studenten anzeigen lassen ✓
- Man kann sich die Adresse des gewählten Studenten anzeigen lassen ✓
- Man kann sich die Telefonnummer des gewählten Studenten anzeigen lassen ✓
- Man kann die Anwendung beenden ✓
- Die Addresse wird in Folgendem Format angegeben: Straße [Leerzeichen] Hausnummer [Neue Zeile] PLZ [Leerzeichen] Ort ✓
- Zur Formatierung existieren die beiden Hilfsklassen Address und PhoneNumber ✓
- Data-Store ist vorhanden ✓
- Einige Unit-Tests sind vorhanden ✗ => Es sind nur 3 Stück vorhanden (nicht ausreichend getestet)
- Die Telefonnummer wird in folgendem Format angegeben: Vorwahl [Schrägstrich] Nummer ✗ => Die Telefonnummer wird nicht durch Schrägstrich sondern Bindestrich getrennt
- Bei der Suche nach einer nicht existenten Matrikelnummer wird eine Fehlermeldung ausgegeben und eine neue Eingabe ermöglicht ✗ => Das Programm crashed

2. Sonstige Probleme

- Bei Eingabe 6 und 7 unerwartete Ausgabe (identisch zu 2 => Ausgabe student.info())
- Bei höheren Eingaben als 8 passiert gar nichts (keine Fehlermeldung oder sonstiges)
- Usability-Problem, ein Student muss ausgewählt werden, für weitere Aktionen, aber das wird dem User nicht kommuniziert
- Country-Code im Datastore wird ignoriert (va bei internationaler Telefonnummer)
- Unit-Tests funktionieren nur teilweise.
    - testFormatInternational ruft die falsche Formatierungs-Funktion auf
    - testFormat => line.seperator ist nicht immer identisch zu \n

3. Benötigte Tests vor Weiterentwicklung

- Test für nicht vorhandene Matrikelnummer
- Test für Wort-Eingabe statt Matrikelnummer
- Test für gültige Matrikelnummer
- Test für Wort-Eingabe bei Menu-Selection
- Test für ungültige Zahleneingabe bei Menu-Selection
- Test für gültige Eingabe mit zuvor ausgewähltem Studenten bei Menu-Selection
- Test für gültige Eingabe (z.B. Adress-Ausgabe) aber nicht ausgewähltem Studenten bei Action-Selection
- Test für Student-Info-Methode
- Alle bestehenden Tests lauffähig machen

4. Weiterentwicklungen

- Formatierung um Frankreich, Großbritannien und USA erweitern (und für Deutschland auslagern)
- Eine Konfiguration des Landes und damit der Formatierung ermöglichen
- Programm um eine Sprachauswahl erweitern
- Die Implementierung so umsetzen, dass es leicht um weitere Länder erweiterbar ist
- Soll-Ist-Differenzen abgleichen, damit alle Soll-Funktionen implementieren (siehe 1.)
- Programm um Resilience erweitern, damit es beispielsweise bei einer Fehleingabe nicht zu einem Crash, sondern eine Fehlermeldung kommt, auf die auch Testbar ist

5. Design Pattern

- Entscheidung: Abstract Factory mit (formatiertem) String als Produkt und die Formatter als konkrete Fabriken (z.B. GermanFormatter, AmericanFormatter), in Kombination mit State-Pattern zum Umschalten des aktuellen Menu-States (z.B. NoLanguageSelected, NoStudentSelected und StudentSelected)
- Entscheidungsgrundlage Abstract Factory:
    - Die Auswahl eines Landes und damit der Formatierung ist bei diesem Ansatz möglich
    - Die Anwendung soll leicht erweiterbar sein. Bei diesem Ansatz muss lediglich ein neuer Formatter, der die Formate über implementierte Funktionen einer Generic-Formatter-Klasse hinzufügt und die Auswahl dieses Formatters im FormatterManager ermöglicht wird
    - Die Konfiguration des Landes kann außerhalb des Quelltextes über z.B. eine Config-Datei, Startparameter oder zur Runtime gesetzte Preferences (oder Properties) gesetzt werden
        => Unsere Entscheidung fiel auf Preferences zur Runtime, weil die Preferences genau zu einem solchen Zweck existieren (Sie persistieren auch)
    - Mischkonfigurationen werden hier per Design ausgeschlossen, da der konkrete Formatter immer nur eine konkrete Factory (eine Sprache) widerspiegelt
- Entscheidungsgrundlage State Pattern:
    - Wechsel zwischen Zuständen im Programm
    - Verschiedene Menus für verschiedene Zustände (zB. Wenn noch kein Student ausgewählt ist, soll auch keine Option zur Ausgabe von student-bezogenen Daten möglich sein/gezeigt werden)
    - Leicht erweiterbar für weitere Menus
- Ausschluss anderer Patterns (die wir im Rahmen der Vorlesung gelernt haben):
    - Singleton => Keinen Nutzen bezüglich Zieles
    - Decorator => War in der Diskussion, da wir aber keine Kombinationen ermöglichen wollen (keine Mischkonfiguration aus Angabe) und nicht mehrere konkrete Decorator (welche in diesem Fall Formatter wären) auf dieselbe „Nachricht“ anwenden wollen, wurde sich dagegen entschieden.
    - Observer => Anderes Anwendungsgebiet. Einziges Anwendungsgebiet, wäre wenn wir über Änderungen am Datastore durch andere Clients benachrichtigt werden wollen, was aber sinnlos ist, da bei jedem Aufruf der Datastore sowieso neu gelesen wird und die Updates daher automatisch berücksichtigt werden
    - Factory Method => Factory Method würde Mischkonfigurationen erlauben, welche nicht gewünscht sind

- Command => Findet keine Anwendung. Andere Anwendungsgebiete
- Facade => Findet keine Anwendung. Andere Anwendungsgebiete
- Adapter => Findet keine Anwendung. Andere Anwendungsgebiete

6. Eigene Gedankengänge

- Aktuell besteht ein Problem, bei Auslandsstudenten/Fernstudium (z.B. ein Amerikaner studiert in Deutschland). Dabei wäre der Wohnort zwar z.B. in Amerika, aber weil per Aufgabenstellung vorausgesetzt wird, dass nur eine Formatierung gleichzeitig unterstützt wird, wäre die Formatierung z.B. Deutsch und damit nicht wie dort üblich. Um diesen Studenten erreichen zu können, wäre eine internationale Anschrift benötigt. Bei Telefonnummern gibt es eine einfache Lösung, weil es ein allgemeines internationales Format gibt, bei Adressen hingegen setzt sich das internationale Format aus dem Format aus dem Zielland (z.B. Amerika) + Name des Ziellands zusammen. Dabei müssten wir wieder auf den jeweiligen Formatter zurückgreifen, was aber direkt im Konflikt mit der Aufgabenstellung steht hinsichtlich, dass nur eine Konfiguration gleichzeitig existieren darf und dies ohne Mischkonfiguration.
- Wir verwenden eine leicht abgeänderte Variante von Abstract Factory, da es sich für diesen Anwendungsfall anbietet. Unsere Abänderung bezieht sich auf die Namensgebung. Dabei haben wir berücksichtigt, dass auch ein Entwickler, der Abstract Factory nicht kennt und die Dokumentation nicht liest sich sehr schnell zurechtfindet. Es ist für jeden Entwickler klar, dass wenn er Formatierungen hinzufügen will, dass er dies im „Formatter“-Ordner tut. Dort findet er bereits mehrere Länder-Formatter, bei welchen er sich einfach einen kopieren, umbenennen und seine Formate implementieren kann. Des Weiteren sollte es klar sein, dass ein Konkreter-Formatter auch immer einen GenericFormatter implementiert (aufgrund der Namensgebung) und dass ein Formatter-Manager diese Verwaltet und man seinen neuen Formatter dort hinzufügen muss. Ein Entwickler der Abstract Factory kennt und die Dokumentation aber nicht liest, wird trotzdem sehr schnell das Pattern erkennen und sich entweder dadurch oder ähnlich wie davor beschrieben über Namensgebung zurechtfinden.

7. Weitere benötigte Unit-Tests

- Tests für jede Sprache/ jedes Format:
  - Test, ob die Info richtig ausgegeben wird
  - Test, ob die Adresse richtig ausgegeben wird
  - Test, ob die Telefonnummer richtig ausgegeben wird
    => Berücksichtigung aller möglichen Pfade. Z.B. hat Großbritannien verschiedene offizielle Telefonnummer-Formate je nach Länge des Area- und Subscriber-Codes. Daher entweder ein Test pro möglichen Pfad und damit möglichem Format (von uns gewählte Methode) oder einen Test, der alle Pfade abdeckt (Unit-Tests, sollen klein und übersichtlich sein und eine Funktion des Programms überprüfen, daher haben wir uns für die Aufteilung in mehrere elementare Tests entschieden)
- Testen der international formatierten Telefonnummer im Generic Formatter (auch hier müssen wie gerade beschrieben alle Pfade getrennt getestet werden)
- Test, auf getFormatter, wenn Sprache noch nicht gesetzt ist
- Test, auf getFormatter, wenn eine nicht bekannte Sprache gesetzt ist
- Test, auf getFormatter, wenn eine bekannte Sprache gesetzt ist
- Test, auf Menu-Übergänge (State-Änderungen)

- Modifikation der Tests von 3. Benötigte Tests vor Weiterentwicklung
- Test, auf gültige Sprach-Auswahl
- Test, auf nicht gültige Sprach-Auswahl

8. Zukünftige Verbesserungen (nicht Teil der aktuellen Anforderungen)

- Berücksichtigung Auslandsstudenten und Fernstudium. Aktuell sind alle Formate wie in dem konfigurierten Land üblich, aber wie bereits beschrieben, kommt es bei Auslandsstudenten zumindest bei der Adresse zu z.B. Deutsch formatiere, amerikanische Adressen. Des Weiteren sollte im gleichen Zug das Mapping von Länderkürzel auf Ländervorwahl (z.B. DE => +49 und GB => +44) auf alle Länder erweitert und evtl. in eine Config-Datei oder ähnliches ausgelagert werden. Beispielsweise kann aktuell ein brasilianischer Student in Amerika studieren, aber es muss nicht zwingend auch unser Programm in Brasilien verwendet werden. Daher sollten zumindest die internationalen Länder-Vorwahlen auf die wichtigsten oder alle erweitert werden.

9. Alternative Lösungen

- Statt der Verwendung von einer Klasse pro Landes-Formatierung, können diese auch als Template in eine externe Textdatei ausgelagert werden. Dadurch lässt sich das Programm, ohne Codekenntnisse um weitere Formatierungen erweitern. Damit wäre auch die internationale Formatierung einfach, weil auf die Formatierung von jedem Land zugegriffen werden kann. Das widerspricht aber den Anforderungen, dass nur eine Sprache zur Runtime gleichzeitig ausgewählt ist und dass hier kein Design-Pattern mehr zu finden ist, wie in Aufgabenpunkt 6 gefordert.