hochschule mannheim

Fakultät für Informatik
Prof. Dr. Peter Knauber

Bachelor-Studiengang Informatik
Übungen zur Vorlesung “Grundlagen der Informatik” (GDI), WS 2011/2012

Übungsblatt 4: Anweisungen und Funktionen

Ausgabe am: 21.10.2011
Abgabe am: 28.10.2011

Aufgabe 1: ISBN-13 Prüfziffer-Berechnung und Validierung — 8 + 8 = 16 Punkte

Schreiben Sie eine Funktion zur Berechnung der Prüfziffer einer ISBN-13 (Internationale Standardbuchnummer). Die Prüfziffer ermöglicht es, eine ISBN-13-Nummer auf ihre Gültigkeit zu überprüfen und so Eingabe- oder Lesefehler zu erkennen. Schauen Sie sich dazu unter folgender URL die Formeln zur Berechnung der Prüfziffer an: http://tinyurl.com/3xovech. Zusätzlich soll eine Prüffunktion geschrieben werden, die eine gültige ISBN-13 Nummer von einer ungültigen unterscheidet.

Verwenden Sie zur Berechnung und Prüfung der ISBN-Nummer folgende Signaturen:

- int computeISBN13CheckDigit(String isbn13)
  Mögliche Eingaben sind ISBN-13 Nummern *ohne* die letzte Prüfziffer; das letzte Trennzeichen darf, muss aber nicht vorhanden sein. Geliefert wird die Prüfziffer für die ISBN oder -1, falls es sich nicht um eine gültige ISBN-13 handelt.
- boolean isISBN13Valid(String isbn13)
  Die Funktion liefert genau dann *true*, wenn es sich um eine gültige ISBN-13-Nummer inklusive Prüfziffer handelt, sonst *false*.

Erstellen Sie diese Funktionen in einer Klasse *ISBNToolkit*, das auch eine *main*-Methode besitzt, in der die Funktionen mithilfe einiger Testfälle gründlich getestet werden.

Hinweise:

- ISBN Nummern können außer Ziffern auch '-' (Trennstriche) oder Leerzeichen enthalten (nach ISO 2108). Entfernen Sie diese am besten vorher oder ignorieren Sie sie. Sie dürfen davon ausgehen, dass das ISBN-13 Format, sprich die Position der Trennzeichen, richtig eingehalten wird.
- Eine ISBN-Nummer enthält entweder genau 12 Ziffern (ohne Prüfziffer) oder 13 Ziffern (inkl. Prüfziffer).

Aufgabe 2 — 10 + 4 + 4 = 18 Punkte

Das S-Bahn-Netz der Stadt Byteburg ist regelmäßig aufgebaut:

- Es gibt fünf Hauptlinien (“1” bis “5”) und eine Ringlinie.
- Die Stationen entlang der Hauptlinien sind von innen nach außen mit 1 bis 6 durchnummeriert. Im Zentrum liegt Station “00”.
- Die Ringlinie verbindet reihum die Stationen 3 der Hauptlinien.
- Die Stationen sind mit einem zweistelligen Code benannt: Die Zehnerstelle nennt die Linie, die Einerstelle die Stationsnummer. In der Skizze unten ist das für zwei Linien eingetragen.
- Linie “6” befindet sich (inklusive Station 63) im Bau und wird noch nicht benutzt.

Im Innenraum (in der Skizze blau hinterlegt) liegen das Zentrum und alle Stationen bis einschließlich der Ringlinie. Die übrigen Stationen liegen im Außenraum. Schreiben Sie ein Programm “ByteburgTarif”, das die Fahrpreise nach den folgenden Regeln berechnet:

- Eine Fahrt kostet 2 Euro (Beispiel: 11 nach 13: 2 Euro).
- Jede überquerte Zonengrenze kostet zusätzlich 1 Euro (Beispiel: 11 nach 14: 3 Euro).
- Jede benutzte Endstation kostet zusätzlich 1 Euro (Beispiel: 16 nach 11: 4 Euro).
- Eine Fahrt zwischen zwei benachbarten Stationen kostet immer 1 Euro, auch wenn eine Zonengrenze dazwischen liegt oder eine Endstation benutzt wird (Beispiele: 13 nach 14, 15 nach 16 und 13 nach 53: jeweils 1 Euro).

Auch nach Bau von Linie “6” bleiben die Hauptlinien im Uhrzeigersinn fortlaufend durchnummeriert. Ihr Programm soll mit möglichst wenig Aufwand für das erweiterte Streckennetz anpassbar sein.

Aufgabenstellung:

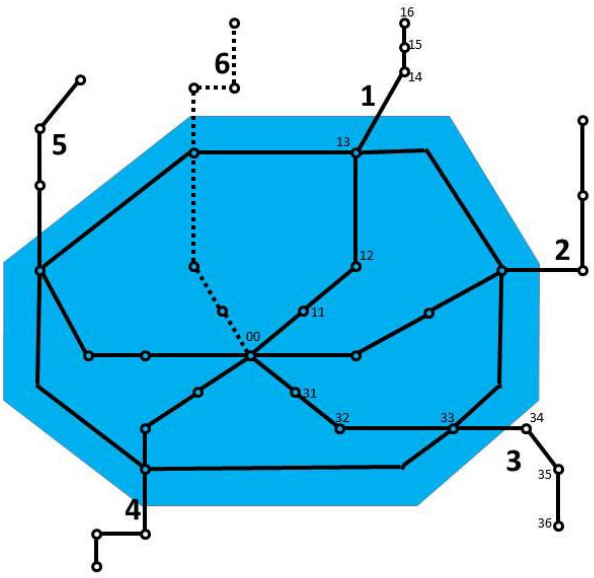
a) Erstellen Sie eine Klasse *ByteburgTarifRechner*, welche die Fahrpreise nach den oben angegebenen Regeln für fünf Linien berechnet. Implementieren Sie in dieser Klasse eine Methode
   int ermittleFahrtkosten(int startStation, int zielStation)
   welche die Fahrtkosten von Station *startStation* nach Station *zielStation* in Euro berechnet.

b) Erstellen Sie in der Klasse *ByteburgTarifRechner* eine *main*-Methode, welche die Codes von zwei Stationen als ganze Zahlen von der Konsole liest und die Fahrtkosten ausgibt. Die eingegebene Stationsnummer soll auf Gültigkeit geprüft werden.

c) Überlegen Sie genau, was Sie an Ihrem Programm ändern müssen, um die Tarife auch für die neue Linie berechnen zu können.

Hinweise

- Die Aufgaben sind in Eclipse zu bearbeiten. Legen Sie, wie in der Vorlesung gezeigt, ein Projekt namens GDI an und binden Sie die Bibliothek gdi11.jar ein. Eine Anleitung dafür finden Sie im Wiki beim Übungsblatt 4.
- Legen Sie für die Bearbeitung dieses Übungsblattes ein Paket (engl. Package) namens *uebung04* an; tun Sie das auch für zukünftige Übungsblätter nach diesem Schema.
- Von allen Aufgaben sind Programmausdrucke (Listings) abzugeben, *keine* Ausdrucke von Testläufen. Die Aufgaben sind im Labor mit Eclipse vorzuführen.
- Erlaubt sind *MakeItSimple*-Funktionen (keine nicht besprochene Funktionalität aus der Java Standard Bibliothek) und das bisher erworbene Wissen aus den GDI-Vorlesungen (keine Java-Konstrukte, die noch nicht behandelt wurden). Zusätzliche eigene Hilfsfunktionen (keine fremden oder externen) sind ausdrücklich erlaubt.
- In den Laborstunden sollen Ihre Programme automatisch getestet werden. Damit Sie vorab prüfen können, ob Ihr jeweiliges Programm äußerlich korrekt ist (was nicht heißt, dass es korrekt funktioniert!), finden Sie im Wiki für jedes Ihrer Programme ein JUnit-Testprogramm. Außerdem finden Sie dort Anleitungen
  - wie JUnit in Ihr GDI-Projekt eingebunden wird und
  - wie Sie das Testprogramm zu einem Ihrer Programme ausführen.



Der blaue Bereich zeigt die Innenzone bzw. den Verlauf der Grenze von der Innen- zur Außenzone,
die Ringlinie als Verbindung der jeweiligen Station 3 hat nichts mit dem Zonenverlauf zu tun.