

Lab02 – File IO

Hinweis: Das Aufgabenblatt enthält neben den Aufgaben zum Modulpraktikum auch Zusatzfragen zum Verständnis und zur Repetition des Stoffes. Diese Fragen müssen nicht für die Fallstudie beantwortet werden und sind freiwillig. Sie können jedoch zur Prüfungsvorbereitung verwendet werden.

Aufgabe 1. Arbeiten mit Git

Milestone Branch einrichten

Die eigentliche Entwicklung der Fallstudie können Sie auf beliebigen Branches in Ihrem Gitlab-Repository durchführen. Die Abgabe der Lösungen geschieht jedoch mittels PullRequest auf den Branch «milestones».

Legen Sie in Ihrem Repo einen neuen Branch «milestones» an.

(Hinweis: Dieser Branch darf NICHT für die Entwicklung verwendet werden. ☹)

Öffnen Sie Ihr Gitlab-Projekt im Browser Ihrer Wahl.

Wählen in der Navigation links die Option «Repository/Branches» und klicken Sie rechts den Button «New branch» und erzeugen Sie den neuen Branch «milestones» (siehe Abb 1).

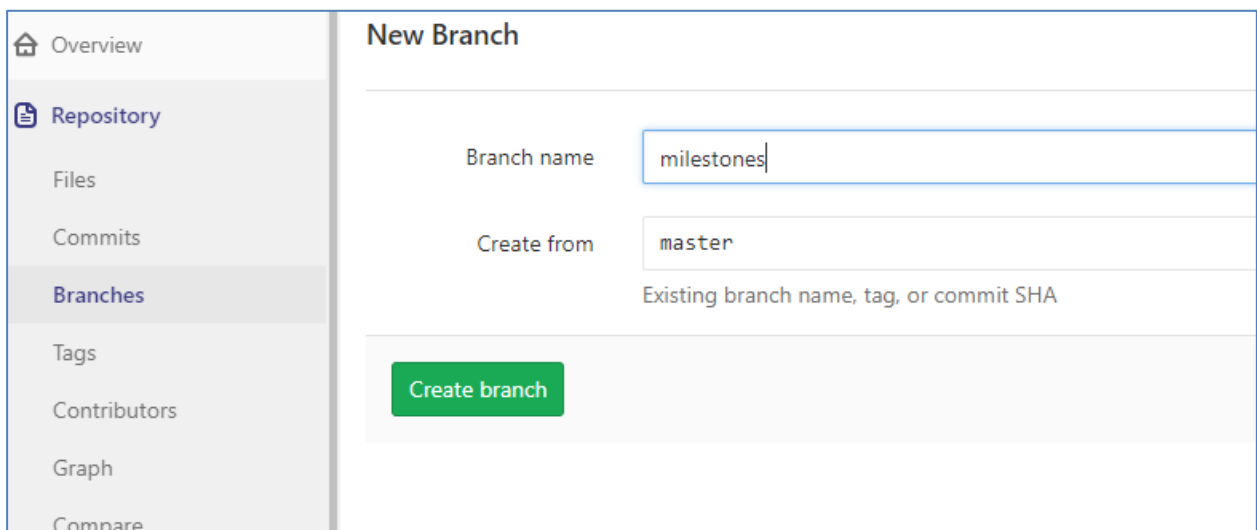


Abbildung 1. Create new branch in gitlab

Aufgabe 2. WayPoint Erweiterung

- a) Die ToString()-Methode der Klasse object erlaubt die Default-String-Konversion einer Klasse zu ändern. Überschreiben Sie ToString() in der Klasse WayPoint so, dass bei der Ausgabe eines WayPoints der folgende Text ausgegeben wird:

"WayPoint: <Name> <Latitude>/<Longitude>"

Die Koordinaten sollen immer auf zwei Nachkommastellen gerundet werden. Falls der Name leer oder null ist, sollen nur die Koordinaten ausgegeben werden.

Bsp:

WayPoint: Windisch 47.54/8.21
WayPoint: 0.54/0.21

Hinweise:

Verwenden Sie zum Runden der Werte Format-String-Interpolation, nicht die Math-Klasse. Verwenden Sie zur Ausgabe die CultureInfo um eine Länderkultur-unabhängige Ausgabe sicherzustellen (Welche Form müssen Sie dafür wählen?).

- b) Ergänzen Sie die Klasse WayPoint durch die Methode double Distance(WayPoint target), welche die Entfernung zum target berechnet, und lassen Sie sich als Test die Entfernung zwischen Bern und Tripolis auf der Konsole ausgeben.

Die Formel für die Berechnung finden Sie in der Beschreibung zur Fallstudie von letzter Woche.

Hinweise:

Achten Sie bei der Berechnung auf die Umrechnung von Winkel in Radiant.

Die Koordinaten der Orte können Sie sich in Google-Maps via rechte Maustaste/"Was ist hier?" anzeigen lassen. Koordinaten in Dezimalen verwenden

Aufgabe 3. File IO, Collections, Indexer

- a) Die Applikation soll die Routen zwischen Städten berechnen. Implementieren Sie nun die Klasse Fhnw.Ecnf.RoutePlanner.RoutePlannerLib.City mit den Properties string Name, string Country, int Population und WayPoint Location. Der Konstruktor soll folgendermassen aufgerufen werden können:

```
new City("Bern", "Schweiz", 75000, 47.479319847061966, 8.212966918945312);
```

- b) Implementieren nun die Klasse Fhnw.Ecnf.RoutePlanner.RoutePlannerLib.Cities, welche mittels der Methode

```
int ReadCities(string filename)
```

die Städtedaten aus der übergebenen Datei einliest und zu einer internen Liste hinzufügt (List<City>). Die Methode soll die Anzahl der neu eingelesenen Städte zurückgeben.

Zu Testzwecken finden Sie dazu auf dem Netzwerkshare die Datei citiesTestDataLab2.txt, die eine Liste von Städten enthält (Tab-getrennt: Name, Land, Einwohner, geografische Breite und Länge):

Berlin	Germany	3274468	52.52	13.38
Montréal	Canada	3247983	45.52	-73.57

Madrid Spain 3169379 40.42 -3.71

Wie Sie mit dem Testfile arbeiten finden Sie in Aufgabe 5 Unit Testing beschrieben.

- c) Fügen Sie weiter die Methode "int AddCity(City city)" hinzu. Die Methode fügt der internen City-Liste die übergebene City hinzu und gibt die neue Anzahl der Cities zurück.
- d) Implementieren Sie für die Klasse Cities das Property Count, die die Anzahl der enthaltenen Cities zurückgibt, sowie einen Indexer, mit dem sich über die enthaltenen Städte iterieren lässt und Direktzugriff auf eine Stadt erlaubt. Beim Zugriff mit einem ungültigen Index soll eine `ArgumentOutOfRangeException` mit einer hilfreichen Message geworfen werden.
- e) Implementieren Sie die Methode

`IList<City> FindNeighbours(WayPoint location, double distance)`

die alle Nachbarstädte von location im Umkreis von distance zurückgibt.

Aufgabe 4. Unit Testing

Sie werden für die weiteren Implementierungen entweder Unit-Tests vorgegeben bekommen, oder selbst weitere Tests schreiben. Zunächst müssen Sie daher Ihrer Solution ein Unit-Test Projekt hinzufügen und konfigurieren.

- a) Fügen Sie mittels „Add/New Project“ (Projekttyp „MSTest Test Project (.NET Core)“) ein neues Projekt mit dem Projektnamen RoutePlannerTest Ihrer Solution hinzu. Visual Studio legt automatisch die Datei UnitTest1.cs an. Löschen Sie diese Datei.
- b) Sie werden im weiteren Verlauf Testdaten (Koordinaten von Städten) aus Dateien lesen. Fügen Sie dazu das Testfile citiesTestDataLab2.txt Ihrem Test-Projekt hinzu. Sie können die Datei über „Add/Existing Item...“ zum Projekt hinzufügen.

Hinweis: Damit das File auch deployed wird, setzen in Visual Studio die File Property „Copy to Output Directory“ auf „Copy if newer“.

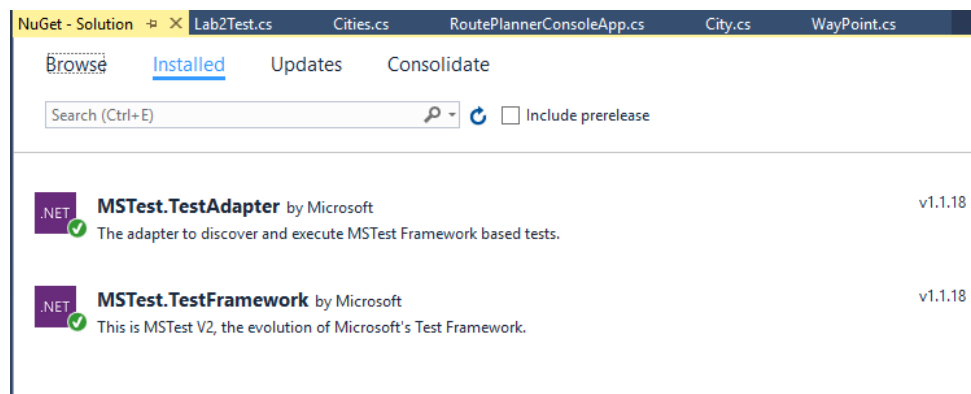
- a) Fügen Sie nun die Dateien *Test_Lab02.cs sowie die Datei «TestHelpers.cs» vom Netzwerkshare dem Test-Projekt hinzu.

Fügen Sie Ihrem Test-Projekt die Referenz auf das RoutePlannerLib Assembly hinzu.

- b) Ergänzen Sie Ihren Code so, dass alle Tests erfolgreich durchlaufen.

Führen Sie nun die Tests über Test/Run/All Tests oder Ctrl-R, A bzw. Test/Debug/All Tests aus.

1. Stellen Sie sicher, dass Sie die neuste Version des MSTest installiert haben.



Testfragen

- Was ist der Hauptunterschied zwischen Structs und Klassen?
- Welche Einschränkungen haben Structs gegenüber Klassen?
- Wann sollten Sie eher Structs, wann eher Klassen einsetzen?
- Demonstrieren Sie den Einsatz und die Effekte von `new`, `override`, `virtual`
- Was ist der Unterschied zwischen `const` und `readonly`? Existieren diese Konzepte in Java?
- Was sind *Named Arguments*?
- Was sind *Optional Arguments*?