Lehrstuhl für Software and Computational Systems
Prof. Dr. Gidon Ernst, Marian Lingsch-Rosenfeld

Übungen zur Vorlesung
Formale Spezifikation und Verifikation

Wintersemester 2024/25

Übungsblatt 06

Bekanntgabe am 02.12.2024

Für diese Aufgabe werden Aufgaben 1 und 2 außer die Bearbeitung der Methoden `Concat` und `ContainsAt` in den Tutorien vorgerechnet.

Die Bearbeitung der Methoden `Concat` und `ContainsAt` in der Klasse `String` sollte in den Tutorien erfolgen.

1 Girokonto (Bankkonto mit separat gelisteter Einzahlung/Auszahlung)

Gegeben ist die Klasse `CheckingAccount` als Unterklasse von `Account` aus Übungsblatt 06. Diese verwaltet mittels zwei Variablen separat die Geldmenge, welche seit dem Anlegen des Kontos eingezahlt wurde sowie jene Geldmenge, die insgesamt abgehoben wurde. Die Klasse `CheckingAccount` erlaubt darüber hinaus auch negative Kontostände (im Gegensatz zu `SavingsAccount` aus der vorigen Teilaufgabe).

Aufgaben

a) Implementieren Sie (in Dafny basierend auf `girokonto.dfy`) die Methoden in folgender Klassendefinition, ohne zusätzliche Attribute hinzuzufügen:

```
class CheckingAccount extends Account {
    // Attribute
    int deposited;
    int withdrawn;

    // Konstruktor
    public CheckingAccount()         { ... }

    // Methoden
    public int getBalance()          { ... }
    public void deposit(int amount)  { ... }
    public void withdraw(int amount) { ... }
}
```

b) Definieren Sie eine korrekte Klasseninvariante, welche die Attribute `deposited` und `withdrawn` mit der/den Modellvariable(n) in Bezug setzt

c) Verifizieren Sie (in Dafny) für den Konstruktor und alle Methoden dass

- Die Klasseninvariante nach den Aufrufen (wieder) etabliert ist.
- Dass die von `Account` geerbten Nachbedingungen gelten

- Geben Sie auch an, welchen neuen Wert die Modellvariable(n) am Ende der Methoden bekommen, also welche Instanzen Sie jeweils für den Existenzquantor im Hoare Tripel gemäß Folie 35 in `09-OOP.pdf` gewählt haben

2 Klasse `String`

Die Klasse `String` implementiert einen String Datentyp mit zwei speziellen Eigenschaften:

- Objekte stellen sich nach außen *unveränderlich* dar, d.h., man kann immer nur neue Strings durch den Konstruktor erzeugen, es gibt aber keine Methode, um bestehende Strings zu verändern.

  Dieses Prinzip für Strings ist grundsätzlich eine gute Idee und wird daher von allen modernen Programmiersprachen verwendet (z.B. Java, Python, Javascript, ...).

- Mehrere String-Objekte können sich das zugrundeliegende Array mit Zeichen teilen, dies ermöglicht eine effiziente Operation um Teilstrings zu erzeugen. Dazu merkt sich jeder String nicht nur das (möglicherweise geteilte) Array sondern auch einen Start- und End-Index der für diesen String relevanten Zeichen.

  Dieses Prinzip der Implementierung von Strings wurde von älteren Java-Versionen genutzt. Allerdings kann dies zu großen Speicherbereichen führen, die nicht mehr freigegeben werden können. Aktuelle Java-Versionen allokieren daher für jedes String-Objekt ein eigenes Array.

Aufgabe:

- Implementieren, Spezifizieren, und Verifizieren Sie die Methoden der Klasse `String`. Dabei ist die bearbeitung der Funktionen `Concat` und `ContainsAt` eine **Präsenzaufgabe** und alle anderen Hausaufgabe.
- Achten Sie darauf, dass in den `requires` und `ensures` Bedingungen der Methoden nur *Parameter*, die Modellvariable `content` (ggfs von unterschiedlichen Strings), sowie das Prädikat `inv` vorkommen. Nur dann ist Ihre Spezifikation unabhängig von der internen Repräsentation.

Hinweise:

- Orientieren Sie sich an den Beispielen **ArrayList** und **GapBuffer** aus der Vorlesung (auf Moodle im Ordner "Dafny Beispiele")
- Überlegen Sie sich zunächst wie `content` mit `chars` und `from,to` zusammenhängt
- Wir empfehlen, mit dem Konstruktor, der Methode `Length`, sowie der Invariante (Prädikat `inv`) anzufangen.
- Ein neues Array von Zeichen können Sie mit `new char[size]` anlegen (keine Notwendigkeit zur Angabe der Initialisierung der Elemente wie bei `ArrayList` in der Vorlesung).
- Die Länge eines Arrays `a` wird mit `a.Length` bestimmt, Die Länge einer mathematischen Sequenz `s` wird dagegen mit `|s|` bestimmt
- `a[from..to]` bezeichnet die Sequenz, welche die Elemente `a[from]` bis `a[to-1]` enthält, wobei $0 \leq$ `from` $\leq$ `to` $\leq$ `a.Length` gelten muss. Ähnlich bei Sequenzen.
- In `Substring` und `ContainsAt` benötigen Sie das lemma `helper`, welches bereits als Kommentar definiert ist. Die Aufrufe des Lemmas sind bereits im Code angegeben und brauchen nicht weiter verändert werden. Sofern Ihre Invariante richtig definiert ist, sollte auch `helper` bewiesen werden können wenn Sie die Kommentarzeichen entfernen.