Praktikum Programmieren
Studiengang Angewandte Informatik
Prof. Dr. Bernd Kahlbrandt
Dipl.-Ing. Gerhard Oelker
Hochschule für Angewandte Wissenschaften Hamburg
Department für Informatik
4. November 2019

Aufgabenblatt A05: Datenstrukturen Queue und Stack

1 Queue

Eine *Queue* ist eine Datenstruktur, bei der am Ende eingefügt und am Anfang entnommen wird, also ein First in First out (FiFo) Speicher.

Implementieren Sie bittein Ruby eine *Queue* mit folgenden Methoden und Eigenschaften unter Verwendung eines Arrays als Datenstruktur für die Elemente!

1. enqueue: Fügt ein neues Elements am Ende ein.
2. dequeue: Entfernt des Elements am Anfang und liefert es zurück.
3. empty?: Liefert *true*, wenn die *Queue* leer ist, andernfalls *false*.
4. Die *Queue* darf kein *nil* enthalten.
5. Die Methode *dequeue* soll eine geeignete *Exception* „raisen“, wenn die *Queue* leer ist. Schreiben Sie dazu bitte eine Unterklasse von *StandardError*.
6. Die Methode *enqueue* soll eine geeignete Exception werfen, wenn versucht wird *nil* einzufügen. Schreiben Sie dazu bitte eine Unterklasse von *StandardError*.
7. Zwei Queues sind gleich, wenn sie die gleichen Elemente in der gleichen Reihenfolge enthalten.
8. Denken Sie bitte auch an die Konsistenz der Methoden aus *Object*, die Sie überschreiben müssen.

Treffen Sie für die folgenden Fragen bitte Entscheidungen:

1. Überlegen Sie sich bitte ob und ggf. welche Parameter die Methoden benötigen und begründen Sie Ihre Entscheidung!
2. Überlegen Sie sich bitte, was bei *enqueue* zurückgeben werden soll und begründen Sie Ihre Entscheidung!

Verwenden Sie **nicht** die Klasse *Queue* aus der Ruby Core Library!

2 Stack

Ein *Stack* ist eine Datenstruktur, bei der am Ende eingefügt und am Ende entnommen wird, also ein Last in First out (LiFo) Speicher.

Implementieren Sie bitte in Ruby einen *Stack* mit folgenden Methoden und Eigenschaften unter Verwendung eines Arrays als Datenstruktur für die Elemente!

1. push: Fügt ein neues Elements am Ende ein.
2. pop: Entfernt des Elements am Ende und liefert es zurück.
3. empty?: Liefert *true*, wenn der *Stack* leer ist, andernfalls *false*.
4. Der *Stack* darf kein *nil* enthalten.
5. Die Methode *pop* soll eine geeignete Exception „raisen“, wenn der *Stack* leer ist. Schreiben Sie dazu bitte eine Unterklasse von *StandardError*.
6. Die Methode *push* soll eine geeignete Exception werfen, wenn versucht wird *nil* einzufügen. Schreiben Sie dazu bitte eine Unterklasse von *StandardError*.
7. Zwei Stacks sind gleich, wenn sie die gleichen Elemente in der gleichen Reihenfolge enthalten.
8. Denken Sie bitte auch an die Konsistenz der Methoden aus *Object*, die Sie überschreiben müssen.

Treffen Sie für die folgenden Fragen bitte Entscheidungen:

1. Überlegen Sie sich bitte ob und ggf. welche Parameter die Methoden benötigen und begründen Sie Ihre Entscheidung!
2. Überlegen Sie sich bitte, was bei *push* und *pop* zurückgeben werden soll und begründen Sie Ihre Entscheidung!

Für beide Aufgabenteile gilt: Schreiben Sie bitte aussagefähige (RUnit-)Testfälle, um zu zeigen, dass Ihre Implementierungen funktionieren!

Hinweis: Um zu testen, ob die Exception korrekt geworfen („raise“) wird, können Sie *assert_raise* verwenden. Der Code — also der Methodenaufruf, der die Exception werfen soll — wird als Block übergeben, die erwartete Exception als Parameter. Mittels *assert_nothing_raised* können Sie testen, ob keine Exception geworfen wird, wenn das nicht geschehen. soll.

Der Abgabetermin für alle ist:

18.11.2019, 12:00

Eingang in meiner Inbox.