W02H02 - Penguin Casino

Hausaufgabe Medium

Points: 5 of 6 Assessment: automatic

Submission due: 5 months ago
Complaint due: 5 months ago

Practice Clone Repository

83.33% (5 months ago) GRADED

Recent results:

0% (6 months ago)	0% (6 months ago)	0% (6 months ago)	0% (5 months ago)	83.33% (5 months ago)
GRADED	GRADED	GRADED	GRADED	GRADED

Show all results

Tasks:

Penguin Casino

Du hast dich nach dem Feierabend auf ein Guinness im allseits bekannten Irish Pub in der Antarktis getroffen. Bei einem ist es anscheinend auch nicht geblieben. Am nächsten Morgen erinnerst du dich, wie ihr euch darüber unterhalten habt, ein Kasino zu eröffnen. Oh je! Als Informatiker*in bist du natürlich für die Software zuständig. Mit einer frischen Tasse Java machst du dich an eine erste Demo-Version ...

In dieser Aufgabe wird eine vereinfachte Version des Glückspiels "Black Jack" implementiert. Halte dich genau an die Anweisungen und achte auch auf die Formatierung bei Ausgaben in der Konsole. Wir empfehlen, die Aufgabe zunächst vollständig zu lesen und erst dann mit der Implementierung zu beginnen. In der Angabe wird zunächst das Gesamtkonzept (Folge von mehreren Spielrunden) beschrieben und dann der Verlauf eines einzelnen Spiels.

Hinweis 1: Auch in dieser Aufgabe wurden alle Leerzeichen, welche von in den Ausgabe-Strings enthalten sein sollen, durch _ ersetzt, um sie sichtbarer zu machen. Dein Programm muss jedoch Leerzeichen und nicht _ ausgeben (siehe Beispiele).
Hinweis 2: Über Overflows musst du dir in dieser Aufgabe keine Gedanken machen, d.h. du kannst davon ausgehen, dass die Zahlenwerte nie bedeutend groß werden.
Hinweis 3: In dieser Aufgabe wirst du eine bereits implementierte Klasse (`CardDeck`) benutzen müssen. Natürlich darf diese von dir nicht verändert werden. Ein Blick in die Datei, um etwas zu lernen, ist trotzdem eine gute Idee.

Gesamtkonzept

Start

Zum Start des Spiels soll der Nutzer mit der folgenden Nachricht in der Konsole begrüßt werden: "`Welcome_to_Pengu-BlackJack!`". Ein Spieler beginnt immer mit `1000` Tokens, die er zum Black Jack spielen verwenden kann. Vor bzw. nach jeder gespielten Runde soll gefragt werden, ob der Nutzer eine weitere Runde spielen möchte oder das Programm beendet werden soll: "`(1)_Start_a_game_or_(2)_exit`". Entsprechend der darauf folgenden Eingabe des Spielers soll dadurch entschieden werden, ob wirklich eine neue Runde gestartet wird oder die Schlusssequenz eingeleitet wird. Einen User-Input kannst du entgegennehmen, in dem du die vorgegebene Methode `readInt()` aufrufst. `int input = readInt()` pausiert das Programm, wartet auf eine Eingabe des Nutzers und legt diese in der Variable `input` ab sobald der Nutzer die Eingabe getätigt hat. Gibt der User den Input `1` will er/sie eine Runde spielen (siehe unten wie), wird eine `2` gelesen, startet die Schlusssequenz. Bei allen anderen möglichen Zahleneingaben soll das Programm sich mit der folgenden Ausgabe über den fehlerhaften Input beschweren: "`What?!`" und den Prozess des Fragens wiederholen.

Schlusssequenz

Die Schlusssequenz besteht aus 3 Ausgaben in der Konsole. In der ersten Zeile soll dem User der finale Kontostand mitgeteilt werden: "`Your_final_balance:_`<#Tokens>". In der darauf Folgenden Zeile wird dem Benutzer angezeigt, ob er nach den gespielten Spielen einen (echt positiven) Gewinn erzielt hat ("`Wohooo!_Ez_profit!_:D`") oder nicht ("`That's_very_very_sad_:(`"). Als letztes wollen wir uns noch von unserem Nutzer verabschieden mit der folgenden Nachricht: "`Thank_you_for_playing._See_you_next_time.`". Mit dieser Nachricht ist das Spiel beendet.

▶ Beispiel 1: Nutzer mit 123 Tokens

▶ Beispiel 2: Nutzer mit 1234 Tokens

Eine einzelne Runde

Entscheidet sich der Nutzer dazu, eine Runde starten zu wollen, muss natürlich zuerst der aktuelle Kontostand ausgegeben werden (zu Beginn `1000`): "`Your_current_balance:_`<#Tokens>". Nun muss der Nutzer seinen Spiel-Betrag wählen können. Dazu muss in der Konsole folgende Frage gestellt werden: "`How_much_do_you_want_to_bet?`". Anschließend kann der User-Input entgegen genommen werden. Dieser Vorgang wird wiederholt, solange die Eingabe nicht strikt positiv ist oder sie den aktuellen Kontostand überschreitet (Frage & Eingabe). Sobald der Betrag feststeht, wird damit begonnen, die Spielerkarten zu ziehen (siehe folgenden Abschnitt).

Spieler Karten

Als erstes möchten wir dem Spieler mitteilen, dass nun er an der Reihe ist. Deshalb starten wir mit folgender Ausgabe: "`Player_cards:`". Nun werden die ersten beiden Karten gezogen. Für jede gezogene Karte möchten wir in der Konsole sowohl ausgeben, die wievielte Karte diese ist, als auch deren Wert ("<Karten-Nr.>`_:_`<Karten-Wert>", *Achtung:* Karten-Nr. starten mit `1`). Um eine Karte vom Stapel (`deck`) zu nehmen musst du die Methode `drawCard()` benutzen: `int card = deck.drawCard()` legt den Wert der gezogenen Karte in die Variable `card`.

Die Werte der Karten sind relevant. Diese werden alle zusammen aufaddiert. Nachdem die ersten `2` Karten gezogen wurden, muss in der Konsole der aktuelle Punktestand ausgegeben werden: "`Current_standing:_`<Punktestand>". Solange die Summe aller Kartenwerte kleiner als `21` hat der User die Wahl eine weitere Karte zu ziehen (`1`) oder den Zug zu beenden (`2`). Dazu generieren wir folgende Ausgabe: "`(1)_draw_another_card_or_(2)_stay`". Wie schon zu Beginn beschrieben soll das Programm auf die Eingabe entsprechend reagieren. Bei einer nicht definierten Eingabe (nicht `1`/`2`) wird auch hier "`What?!`" ausgegeben und der Process (Frage&Input) wiederholt.

Entscheidet sich der Spieler dazu eine weitere Karte zu ziehen, soll das Programm dieses Umsetzen und sowohl die gezogene Karte ("<Karten-Nr.>`_:_`<Karten-Wert>", *Achtung:* fortlaufende Indizes) als auch den danach aufsummierten Punktestand ("`Current_standing:_`<Punktestand>") ausgeben. Dieser Vorgang wird solange wiederholt, bis der User sich gegen eine weitere Karte entscheidet oder die Punktzahl `20` Punkte überschreitet.

Nach dem der Benutzer seinen Zug beendet hat, kann es drei mögliche Szenarien geben.

1. *Punktzahl `21` überschritten:* Der Spieler verliert das Spiel. Dazu geben wir folgenden String in der Konsole aus: "`You_lost_`<Spiel-Betrag>`_tokens.`". Der Kontostand muss dementsprechend angepasst werden.
2. *Punktzahl `21` (Black Jack):* Der Spieler gewinnt den Jackpot (das Doppelte des Spiel-Betrages). Dazu geben wir folgenden String in der Konsole aus: "`Blackjack!_You_won_`<doppelter Spiel-Betrag>`_tokens.`". Der Kontostand muss dementsprechend angepasst werden.
3. *Weniger als `21` Punkte:* Hier wird es nun etwas schwieriger. Denn jetzt muss der Dealer spielen. Dieser zieht solange Karten, bis er entweder den Punktestand des Spielers überbietet oder bis er selbst einen Punktestand von `21` überschreitet. Zunächst soll dazu die Überschrift "`Dealer_cards:`" ausgegeben werden. Für jede gezogene Karte soll auch hier die Karte wie schon zuvor ausgegeben werden: "<Karten-Nr.>`_:_`<Karten-Wert>", *Achtung:* Indizes starten erneut bei 1. Nach dem der Dealer seine Karten gezogen hat, soll noch der Punktestand des Dealers ausgegeben werden: "`Dealer:_`<Punktestand-Dealer>". Letztendlich muss noch entschieden werden, ob der Spieler gegen den Dealer gewonnen hat oder nicht. Überschreitet der Punktestand des Dealers `21`, so gewinnt der Spieler und erhält die Ausgabe "`You_won_`<Spiel-Betrag>`_tokens.`". Andernfalls hat der Spieler die Runde und damit auch seine Tokens verloren. Verliert der Spieler, geben wir folgenden String in der Konsole aus: "`Dealer_wins._You_lost_`<Spiel-Betrag>`_tokens.`". In beiden Fällen muss der Kontostand entsprechend angepasst werden: Wenn der Spieler gegen den Dealer gewinnt, wird der Spiel-Betrag zum ursprünglichen Kontostand hinzuaddiert. Wenn der Spieler gegen den Dealer verliert, wird der Spiel-Betrag vom ursprünglichen Kontostand abgezogen.

Eigentlich ist damit eine Runde beendet. Wir wollen aber nicht, dass ein Spieler ohne Tokens weiter spielen kann. Hat der Spieler nach einer Runde keine Tokens mehr, dann ist das Spiel für sie/ihn vorbei. In diesem Fall soll nicht gefragt werden, ob er/sie weiterspielen will, sondern direkt der Satz "`Sorry,_you_are_broke._Better_Luck_next_time.`" auf der Konsole ausgegeben werden. Anschließend soll die bekannte Schlusssequenz folgen und das Programm beendet werden. Andernfalls gelangt der Spieler zurück an den Anfang und wird wiederholt gefragt wie sie/er nun fortfahren möchte (siehe Gesamtkonzept oben).

Seeds, ein ausführliches Beispiel & Tests

Seeds

Statt wie im Template vorgegeben `CardDeck deck = CardDeck.getDeck()` zu benutzen um einen Kartenstapel zu nehmen, kannst du der Methode auch einen Parameter übergeben `CardDeck deck = CardDeck.getDeck(420)`. Die Zahl wird als Seed für die Zufallsgenerierung benutzt, d.h. bei zwei (oder beliebig vielen) Durchläufen, die mit dem selben Seed starten, werden die selben Karten gezogen. So kannst du das folgende Beispiel testen oder selbst eigene Tests generieren. *ACHTUNG:* den Seed sollst du nur zum Testen verwenden, nicht aber für die finale Abgabe.

Beispiel

▶ Beispiel: Drei Runden in Folge (Seed: 420)

Tests

Hier werden dir die Ergebnisse der automatischen Tests direkt angezeigt:

✓ **Öffentliche Tests** 3 of 3 tests passing
Testet deine Abgabe vor der Deadline.

✗ **Versteckte Tests** 12 of 13 tests passing
Testet deine Abgabe nach der Deadline.

✗ **Grading Tests** 5 of 6 tests passing
Testet deine Abgabe nach der Deadline und verteilt Punkte.

Viel Erfolg!

Lösungsvorschlag

Tests

Ein Paar Tage nach der Deadline werden diese beiden Links zu der Musterlösung bzw. unseren PublicTests und HiddenTests führen.

Every student is allowed to complain once per exercise. In total 1000 complaints are possible in this course. You still have 998 complaints left.

Complain