Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Architecture-driven Requirements Engineering (ARE)
https://sdqweb.ipd.kit.edu/wiki/Programmieren
programmieren-vorlesung@ipd.kit.edu

Programmieren – Wintersemester 2019/20

Abschlussaufgabe 2 20 Punkte
Version 1

Ausgabe:	24.02.2020, ca. 13:00 Uhr
Praktomat:	09.03.2020, 13:00 Uhr
Abgabefrist:	24.03.2020, 06:00 Uhr

Abgabehinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der öffentlichen Tests für eine erfolgreiche Abgabe dieses Blattes notwendig ist. Der Praktomat wird Ihre Abgabe zurückweisen, falls eine der nachfolgenden Regeln verletzt ist. Eine zurückgewiesene Abgabe wird automatisch mit null Punkten bewertet. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie keine Elemente der Java-Bibliotheken, ausgenommen Elemente der Pakete `java.lang`, `java.util`, `java.util.regex`, `java.util.function` und `java.util.stream`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 120 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute.
- Nutzen Sie nicht das `default`-Package.
- `System.exit()` und `Runtime.exit()` dürfen nicht verwendet werden.
- Halten Sie die Regeln zur Javadoc-Dokumentation ein.
- Halten Sie auch alle anderen Checkstyle-Regeln an.

Bearbeitungshinweise

Diese Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe, jedoch wird der Praktomat Ihre Abgabe **nicht** zurückweisen, falls eine der nachfolgenden Regeln verletzt ist.

- Beachten Sie, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im ILIAS-Wiki ein.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.
- Die Kommentare sollen einheitlich in englischer oder deutscher Sprache verfasst werden.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

Plagiat

Es werden nur selbstständig erarbeitete Lösungen akzeptiert. Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen wie beispielsweise der Lösungsvorschläge des Übungsbetriebes, ist ein Täuschungsversuch und führt zur Bewertung „nicht bestanden“. Ausdrücklich ausgenommen hiervon sind Quelltextschnipsel von den Vorlesungsfolien. Alle benutzten Hilfsmittel müssen vollständig und genau angegeben werden und alles was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, muss deutlich kenntlich gemacht werden. Für weitere Ausführungen sei auf die Einverständniserklärung (Disclaimer) verwiesen.

Checkstyle

Der Praktomat überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen der Praktomat die Abgabe zurückweist, da diese Regel verpflichtend einzuhalten ist. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki (ILIAS) beschreibt, wie das funktioniert.

>_ Terminal-Klasse

Laden Sie für diese Aufgabe die Terminal-Klasse herunter und platzieren Sie diese unbedingt im Paket `edu.kit.informatik`. Die Methode `Terminal.readLine()` liest eine Benutzereingabe von der Konsole und ersetzt `System.in`. Die Methode `Terminal.printLine()` schreibt eine Ausgabe auf die Konsole und ersetzt `System.out`. Verwenden Sie für jegliche Konsoleneingabe oder Konsolenausgabe die Terminal-Klasse. Verwenden Sie in keinem Fall `System.in` oder `System.out`. Fehlermeldungen werden ausschließlich über die Terminal-Klasse ausgegeben und müssen aus technischen Gründen unbedingt mit `Error,␣` beginnen. Dies kann unter anderem bequem über die Terminal.printError-Methode erfolgen. Laden Sie die Terminal-Klasse **niemals** zusammen mit Ihrer Abgabe hoch.

Kartenspiel (20 Punkte)

In dieser Abschlussaufgabe soll ein Kartenspiel implementiert werden, bei dem ein Spieler versucht auf einer einsamen Insel zu überleben und von ihr zu flüchten. Dazu zieht er von einem Stapel Karten und sammelt somit u.a. Ressourcen, mit denen er Gegenstände bauen kann, die ihn vor seinen Feinden schützen oder die ihm eine Rettung von der Insel ermöglichen. Neben den insgesamt 64 Spielkarten kommen in unterschiedlichen Spielphasen (s.u.) drei Würfel (4-, 6-, und 8-seitig) zum Einsatz.

Jede Spielkarte kann zu einer der insgesamt drei Kategorie zugeordnet werden:
(i) Ressourcen (`resources`), (ii) feindliche Tiere (`animals`), (iii) Katastrophe (`catastrophe`).

Die insgesamt 64 Spielkarten ergeben sich wie folgt:

- Ressourcen
  - 16 × Holz `(wood)`
  - 16 × Metall `(metal)`
  - 16 × Plastik `(plastic)`
- Tiere
  - 5 × Spinne `(spider)`
  - 5 × Schlange `(snake)`
  - 5 × Tiger `(tiger)`
- Katastrophe
  - 1 × Gewitter `(thunderstorm)`

Zu den Ressourcen zählen, wie oben aufgelistet, Holz (`wood`), Metall (`metal`) und Plastik (`plastic`). Mit einer bestimmten Kombination an Ressourcen ist es möglich, (a) Werkzeuge zu bauen, (b) Bauwerke zu errichten und (c) Rettungen zu initiieren. Im Folgendem werden die Ressourcen-Kombinationen den baubaren Gegenstände zugeordnet.

- Werkzeuge
  - Axt (`axe`): 3 × Metall
  - Keule (`club`) 3 × Holz
- Bauwerke
  - Hütte (`shack`): 2 × Holz, 1 × Metall, 2 × Plastik
  - Feuerstelle (`fireplace`): 3 × Holz, 1 × Metal
- Rettungen
  - Segelboot (`sailingraft`): 4 × Holz, 2 × Metall, 2 × Plastik
  - Hanggleiter (`hangglider`): 2 × Holz, 2 × Metall, 4 × Plastik

- Dampfschiff (`steamboat`): 6 × Metall, 1 × Plastik
- Heißluftballon (`ballon`): 1 × Holz, 6 × Plastik

Der Spieler zieht während des Spiels immer die oberste Karten von einem Stapel und kann wie bereits beschrieben Ressourcen sammeln bzw. mit ihnen Gegenstände bauen und auf Tiere treffen, die ihm gesammelte Ressourcen kosten können.

Das Spiel endet, wenn sich der Spieler von der Insel retten konnte oder nach dem Zug der letzten Karte keine Aktion mehr möglich ist.

Das Bauen der Rettungsgegenstände Dampfschiff (`steamboat`) und Heißluftballon (`ballon`) sind nur mit einer bereits gebauten Feuerstelle (`fireplace`) möglich. Weitere Rettungsgegenstände müssen beim Bauen die oben angegebenen Bedingungen erfüllen. D.h. die benötigten Ressourcen müssen vor dem Bauen im Besitz des Spielers sein; die Ressourcen-Karten werden nach dem Bau aus dem Spiel genommen.

Zieht der Spieler eine Karte aus der Kategorie Tiere (`animals`) muss er würfeln und verliert gegebenenfalls alle seine Ressourcen. Ausgenommen sind fünf Ressourcen, die in seiner gebauten Hütte zuvor in Sicherheit gebrachte wurden. Hierbei verwendet er den 4-seitigen Würfel bei der Spinne, den 6-seitigen bei der Schlange und den 8-seitigen bei dem Tiger. Um zu gewinnen und seine Ressourcen zu retten, muss der Spieler bei einer Spinne eine Augenzahl echt größer 2, bei einer Schlange eine Augenzahl echt größer 3, bei einem Tiger eine Augenzahl echt größer 4 würfeln. Zusätzlich zu der Augenzahl kann es einen Bonus von 2 (beim Besitz einer Axt) oder 1 (beim Besitz einer Keule) geben, der zu der gewürfelten Augenzahl hinzu addiert wird. Hierbei darf nur ein Werkzeug (Axt oder Keule) verwendet werden.

Zieht der Spieler die Karte aus der Kategorie Katastrophe (`catastrophe`), so verliert er seine Feuerstelle und alle seine gesammelten und bereits gezogenen Karten (Ressourcen). Ausgenommen sind die in der Hütte (`shack`) gesicherten fünf Ressourcen.

Spielphasen

Bei diesem Kartenspiel werden unterschiedliche Spielphasen definiert sowie die Zulässigkeit der unten beschriebenen Befehle festgelegt. Abbildung 0.1 illustriert diesen Sachverhalt.

Zu Beginn des Spiels hat der Spieler die Möglichkeit, Karten zu ziehen und unter den oben genannten Bedingungen Gegenstände zu bauen (*Scavenge*). Ist ein Gegenstand in der Kategorie Rettungen gebaut worden, so muss in der Regel gewürfelt werden, sodass entscheiden werden kann, ob eine Rettung erfolgt (*Endeavor*). Bei dem Bau eines Segelboots (`sailingraft`) und eines Hanggleiters (`hangglider`) ist eine Rettung garantiert, wenn nach dem Bauen mit dem 6-seitigen Würfel eine Augenzahl größer gleich 4 gewürfelt wird (*End*). Schlägt der Rettungsversuch fehl, so kann der Spieler erneut Karten ziehen und Gegenstände bauen (*Scavenge*), wenn noch Karten im Stapel vorhanden sind. Das Bauen eines Dampfschiffes und eines Heißluftballons ist nur mit dem Besitz einer Feuerstelle möglich. Durch die gegebene garantierte Rettung ist ein Würfeln nicht nötig (*End*). Wenn ein Tier gezogen wurde, muss der Spieler gegen ihn kämpfen (*Encounter*). Die oben angegebenen Regeln entscheiden über Sieg oder Niederlage.

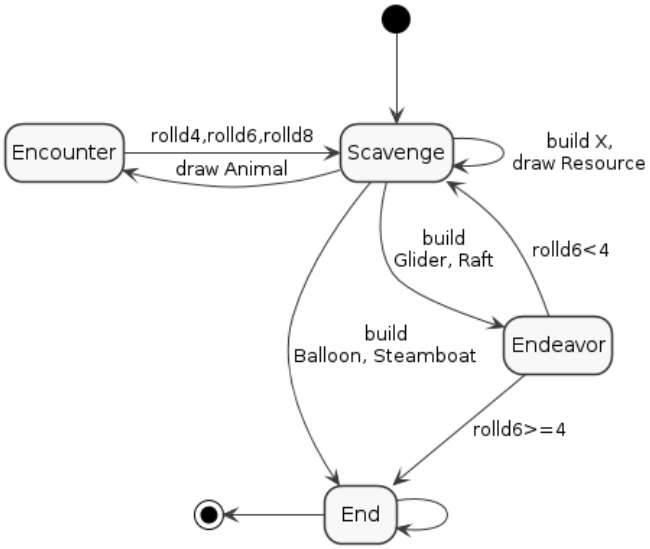

Abbildung 0.1: Spielzustände

Das Ende des Spiels (*End*) ist durch eine garantierte Rettung gegeben oder wenn nach dem Zug der letzten Karte vom Stapel keine weitere Aktion möglich ist.

Interaktive Benutzerschnittstelle

Nach dem Start nimmt Ihr Programm über die Konsole mittels `Terminal.readLine()` verschiedene Arten von Befehlen entgegen, die im Folgenden näher spezifiziert werden. Nach Abarbeitung eines Befehls wartet Ihr Programm auf weitere Befehle, bis das Programm irgendwann durch den `quit`-Befehl beendet wird. Alle Befehle werden auf dem aktuellen Zustand des Spiels ausgeführt.

Achten Sie darauf, dass durch Ausführung der folgenden Befehle die zuvor definierten Regeln nicht verletzt werden und geben Sie in diesen Fällen immer eine aussagekräftige Fehlermeldung aus.

Entspricht eine Eingabe nicht dem vorgegebenen Format, dann ist **immer** eine Fehlermeldung auszugeben. Danach soll das Programm auf die nächste Eingabe warten. Bei Fehlermeldungen dürfen Sie den Text frei wählen, er sollte jedoch sinnvoll sein. Jede Fehlermeldung **muss** mit `Error,` beginnen und darf keine Zeilenumbrüche enthalten.

Da bei der Abgabe automatische Tests Ihrer interaktiven Benutzerschnittstelle durchführt werden, müssen die Ausgaben exakt den Vorgaben entsprechen. Insbesondere sollen sowohl Klein- und Großbuchstaben als auch die Leerzeichen und Zeilenumbrüche genau übereinstimmen. Geben Sie auch **keine** zusätzlichen Informationen aus. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären.

Beachten Sie, dass bei der Beschreibung der Eingabe- und Ausgabeformate die Wörter zwischen spitzen Klammen (`<` und `>`) für Platzhalter stehen, welche bei der konkreten Ein- und Ausgabe durch Werte ersetzt werden. Diese eigentlichen Werte enthalten bei der Ein- und Ausgabe keine Klammern.

Der `start`-Befehl

Dieser Befehl ermöglicht es dem Benutzer, ein neues Spiel zu starten. Dieser Befehl kann nur dann ausgeführt werden, wenn kein Spiel aktiv ist. Zu jedem Zeitpunkt soll höchstens (den Regeln entsprechend) ein Spiel aktiv sein. Zum Programmstart ist kein Spiel aktiv.

Es gibt einen Kartenstapel, von dem der Spieler sukzessiv Karten ziehen kann. Hierbei wird die Reihenfolge der 64 Spielkarten im Stapel festgelegt beginnend mit der oberen aufgelegten Karte. Es werden die englischsprachigen Bezeichner der Spielkarten, wie oben angegeben, verwendet. Die sieben Spielkartenbezeichner (`wood`, `metal`, `plastic`, `spider`, `snake`, `tiger`, `thunderstorm`) werden bei der Eingabe durch exakt ein Komma getrennt. `<cards>` repräsentiert in dem hier angegebenen Eingabeformat einen Platzhalter.

Eingabeformat

```
start <cards>
```

Ausgabeformat

Falls die Parameter den Spezifikationen entsprochen haben und den Regel entsprechend ein neues Kartenspiel gestartet werden konnte, wird `OK` ausgegeben. Falls ein oder mehrere falsche Parameter dem Befehl übergeben wurden, wird kein Spiel gestartet und nur eine Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `draw`-Befehl

Der parameterlose `draw`-Befehl ermöglicht dem Spieler die oberste Karte vom Stapeln zu ziehen.

Eingabeformat

```
draw
```

Ausgabeformat

Es existieren folgende Fallunterscheidungen:

- Wenn sich keine Karte mehr auf dem Stapel befindet bzw. der Befehl in einer nicht für den Befehl vorhergesehenen Spielphase durchgeführt wird, ist dieser Spielzug nicht zulässig. Entsprechend wird eine Fehlermeldung ausgegeben.
- Zieht der Spieler eine Karte, so wird der Name der Ressource bzw. der englischsprachige Bezeichner aus den o.g. Kategorien (i) Ressourcen, (ii) Tiere, (iii) Katastrophe ausgegeben. Entsprechend der gezogenen Karte kann sich die Spielphase ändern.

Beispielablauf: `draw`-Befehl

```
> draw a
Error, incorrect input format.
> draw
wood
> draw
snake
```

Der `list-resources`-Befehl

Der parameterlose `list-resources`-Befehl listet die bereits gezogenen Karten des Spielers auf. Er kann in jedem Spielzustand bzw. -phase durchgeführt werden.

Eingabeformat

`list-resources`

Ausgabeformat

Im Erfolgsfall werden alle aktuell im Besitz befindlichen Ressourcen zeilenweise ausgegeben. Die Ausgabe ist geordnet absteigend nach dem Aufnahmezeitpunkt der Ressourcen. Folglich sind die obersten fünf Ressourcen die, die in der Hütte gesichert werden. Besitzt der Spieler keine Ressourcen, wird `EMPTY` ausgegeben.

Im Fehlerfall (z.B. bei einem fehlerhaften Eingabeformat) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Beispielablauf: `list-resources`-Befehl

```
> list-resources a
Error, incorrect input format.
> list-resources
wood
metal
wood
plastic
```

Der `build`-Befehl

Der `build`-Befehl im Spielzustand *Scavenge* ermöglicht dem Spieler, wie oben angegeben, Gegenstände aus den Kategorien Werkzeuge, Bauwerke und Rettungen zu bauen. `<x>` ist ein Platzhalter der englischsprachigen Bezeichner dieser Gegenstände. Zum Bau dieser Gegenstände werden immer die zuletzt gezogenen Ressourcen priorisiert verwendet.

Eingabeformat

`build <x>`

Ausgabeformat

Im Erfolgsfall (d.h. der Spieler erfüllt die Voraussetzungen bzw. besitzt die jeweiligen Karten zum Bauen der Gegenstände) wird `OK` ausgegeben. Bei erfolgreicher Durchführung werden die eingesetzten Karten zum Bauen aus dem Spiel genommen.

Bei dem Besitz einer Hütte werden automatisch immer die letzten fünf gezogenen Ressourcen gesichert. Wenn diese Ressourcen zum Bauen verwendet werden, rücken die verbleibenden Ressourcen auf.

Im Fehlerfall (z.B., wenn das Bauen durch das Fehlen der Ressourcenkarten nicht möglich ist, bei fehlerhaftem Eingabeformat usw.) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Beispielablauf: `build`-Befehl

```
> list-resources
metal
metal
metal
wood
> build axe
OK
> list-resources
wood
```

Der `list-buildings`-Befehl

Der parameterlose `list-buildings`-Befehl listet die gebauten Gegenstände auf, die sich aktuell noch im Besitz des Spielers befinden. Er kann in jedem Spielzustand durchgeführt werden.

Eingabeformat

```
list-buildings
```

Ausgabeformat

Im Erfolgsfall wird eine Liste der gebauten Gegenstände des Spielers nach Bauzeitpunkt zeilenweise ausgegeben, d.h. das erste Element der Ausgabe ist damit der letzte gebaute Gegenstand. Hat der Spieler noch keine Gegenstände gebaut, wird `EMPTY` ausgegeben. Im Fehlerfall (z.B. bei einem fehlerhaften Eingabeformat usw.) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `build?`-Befehl

Mit dem parameterlosen `build?`-Befehl werden alle möglichen baubaren Gegenständen ausgegeben, die der Spieler mit seinen gezogenen Ressourcen bauen kann.

Eingabeformat

```
build?
```

Ausgabeformat

Im Erfolgsfall werden die potentiell zu bauenden Gegenstände zeilenweise nach Namen alphabetisch aufsteigend sortiert ausgegeben. Ist es nicht möglich Gegenstände zu bauen, wird `EMPTY` ausgegeben. Im Fehlerfall (z.B. bei einem fehlerhaften Eingabeformat usw.) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Der `rollDx`-Befehl

Mit dem `rollDx`-Befehl kann man entweder einen 4-, 6- oder 8-seitigen Würfel auswählen und würfeln, d.h. $x \in \{4, 6, 8\}$. Der Platzhalter `<y>` im Eingabeformat steht für die jeweilige Augenzahl, die gewürfelt wurde.

Eingabeformat

```
rollDx <y>
```

Ausgabeformat

Für das Ausgabeformat ergeben sich im Falle eines Gewinns folgende Fallunterscheidungen:

- Bei dem Kampf gegen Tiere (beim Ziehen einer Karte aus der Kategorie `animals`), wird `survived` ausgeben.
- Beim Bau der Rettungsgegenstände (Segelboot und Hanggleiter) wird `win` ausgegeben.

Ist die Augenzahl kleiner als der jeweilige Schwellwert (s.o.), wird `lose` ausgegeben.

Im Fehlerfall (z.B., wenn der Befehl nicht in der richtigen Spielphase verwendet wird, bei fehlerhaftem Eingabeformat usw.) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Beispielablauf: `rollDx`-Befehl

```
> draw
snake
> rollD6 a
Error, incorrect input format.
> rollD4 4
Error, wrong dice.
> rollD6 6
survived
```

Der `reset`-Befehl

Der parameterlose `reset`-Befehl ermöglicht es, das Kartenspiel neu zu initialisieren. Dieser Befehl kann in jeder Spielphase durchgeführt werden.

Eingabeformat

`reset`

Ausgabeformat

Im Erfolgsfall wird `OK` ausgegeben. Im Fehlerfall (z.B. bei einem falsch spezifizierten Eingabeformat) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Beispielablauf: `reset`-Befehl

```
> reset a
Error, incorrect input format.
> reset
OK
```

Der `quit`-Befehl

Der parameterlose `quit`-Befehl ermöglicht es, das laufende Programm vollständig zu beenden. Beachten Sie, dass hierfür keine Methoden wie `System.exit()` oder `Runtime.exit()` verwendet werden dürfen.

Eingabeformat

```
quit
```

Ausgabeformat

Im Erfolgsfall findet keine Ausgabe statt. Im Fehlerfall (z.B. bei einem falsch spezifizierten Eingabeformat) wird eine aussagekräftige Fehlermeldung beginnend mit `Error,` ausgegeben.

Beispielablauf: `quit`-Befehl

```
> quit quit
Error, incorrect input format.
> quit
```