Projektblatt P 9 (22 P)

Abgabe: Donnerstag 14. November 2024, 12:00h

Entpacken Sie zunächst die Archiv-Datei `vorgaben-p9.zip`, in der sich die Rahmendateien für die Aufgaben 1 und 2 befinden. Ergänzen Sie die Rahmendateien zunächst durch Ihren Namen. Ergänzen Sie die Dateien dann durch Ihre Lösungen gemäß der Aufgabenstellung unten. Der hinzuzufügende Java-Sourcecode sollte **syntaktisch richtig** und **vollständig formatiert** sein. Alle Java-Dateien sollten am Ende fehlerfrei übersetzt werden können. Verpacken Sie die von Ihnen bearbeiteten `.java` Dateien für Ihre Abgabe in einem ZIP-Archiv mit dem Namen `IhrNachname.IhrVorname.P9.zip`, welches Sie auf Ilias hochladen.
Führen Sie dazu in dem Verzeichnis, in dem Sie die Dateien bearbeitet haben, folgenden Befehl auf der Kommandozeile aus:

```
zip  IhrNachname.IhrVorname.P9.zip  *.java
```

Nutzen Sie beim Bearbeiten der Aufgaben nur die Klassen und Methoden aus der Java API, welche explizit durch den Aufgabentext erlaubt sind.

Aufgabe 1: Abstrakte Klassen 15 (= 12+3) Punkte

In diesem Spiel sollen Sie eine Kommandozeilen-Variante des Spiels "*Vier gewinnt* " entwickeln. Die Regeln dieses Spiels können Sie bei Bedarf auf der Seite `https://de.wikipedia.org/wiki/Vier_gewinnt` nachlesen. Für die Implementierung sollen Sie dabei auf verschiedene vorgegebene (abstrakte) Klassen zurückgreifen.

- Die Klasse `Spieler` stellt Funktionalität für einen (von mehreren) Spielern in einem (beliebigen) Spiel zur Verfügung. Sehen Sie sich hierzu die Kommentare in der Datei `Spieler.java` an.

- Die Klasse `Spielstein` repräsentiert einen einzelnen Spielstein, welcher einem von maximal vier möglichen Spielern gehören kann. Jedem Spielstein ist ein Zeichen zur Darstellung des Spielsteins auf der Kommandozeile zugeordnet sowie eine Spielfarbe (mit einem Wert aus der Klasse `java.awt.Color`), die in dieser Aufgabe allerdings nicht benötigt wird. Neben einem Konstruktor bietet diese Klasse hauptsächlich Zugriffsmethoden auf die Instanzen-Attribute der Klasse an, sowie eine Methode, um zu überprüfen, ob zwei Spielsteine dem gleichen Spieler gehören. Schauen Sie sich auch hierzu die Kommentare in der Datei `Spielstein.java` an.

- Die abstrakte Klasse `Spielbrett` repräsentiert ein Spielfeld, welches durch eine regelmäßige Anordnung von Feldern in m Zeilen und n Spalten dargestellt werden kann (wie es z.B. bei einem Schachbrett der Fall ist). Hierzu wird ein zweidimensionales Feld vom Typ Spielstein als Attribut zur Verfügung gestellt und durch den Konstruktor initialisiert. Darüberhinaus dient die vorgegebenen Methode `zeichnen` dazu, das Spielfeld mit dem Inhalt auf der Kommandozeile auszugeben. Leere Felder werden dabei mit einem Unterstrich ('_') dargestellt. Die abstrakten Methoden `zugMoeglich`, `gewinnsituation`, `hinzufuegen` definieren typische Aktionen auf einem Spielfeld, die jedoch spielabhängig implementiert werden müssen. Die Methoden `entfernen` und `bewegen` definieren weitere solche Aktionen, die allerdings im Folgenden nicht benötigt werden.

- Die abstrakte Klasse `Spiel` repräsentiert ein einfaches Brettspiel (auf einem schachbrettartigen Spielfeld) und stellt hierfür einen Konstruktor

zur Verfügung, der das Spiel mit Hilfe der Parameter mit einer vorgegebenen Anzahl von Spielern mit einer ebenfalls vorgegebenen Anazhl von Spielsteinen sowie mit einem Spielbrett ausstattet. Darüberhinaus ist eine Methode `spielen` vorgegeben, die den Spielverlauf implementiert, in dem die alle Spieler der Reihe nach solange ziehen, bis eine Gewinnsituation auf dem Brett eingetreten ist oder kein Zug mehr möglich ist (für keinen der Spieler in jeweils einer Runde). Nach jedem Zug wird dazu das Spielfeld im aktuellen Zustand gezeichnet. Wie ein einzelner Zug eines Spielers ausgeführt wird, muß durch Implementierung der Methode `ziehen` in einer konkreten Unterklasse von `Spiel` festgelegt werden.

Im Folgenden sollen Sie mit Hilfe der Vorgaben das konkrete Spiel "*Vier Gewinnt*" implementieren. Gehen Sie dabei wir folgt vor:

(a) Leiten Sie eine Klasse `VierGewinntBrett` von der Klasse `Spielbrett` ab und implementieren sie einen passenden Kontruktor mit zwei Parametern für die Anzahl der Zeilen und Spalten des Spielbretts, der explizit den Konstruktor der Oberklasse verwendet um die Größe des Spielbretts zu initialisieren Implementieren Sie die Methoden `zugMoeglich`, `gewinnsituation` und `hinzufuegen` gemäß der folgenden "*Vier Gewinnt*" Regeln:

- Unabhängig vom Spieler ist ein Zug immer dann noch möglich, wenn mindestens ein freies Feld (ohne Belegung durch einen Spielstein) auf dem Spielfeld ist.
- Eine Gewinnsituation liegt vor, wenn horizontal, vertikal oder diagonal vier unmittelbar aufeinanderfolgenden Steine die gleiche Farbe haben, also dem gleichen Spieler gehören.
- Ein Hinzufügen eines Spielsteins für einen Spieler in einer Spalte (die Zeile wird nicht verwendet), ist genau dann möglich, wenn die Spaltennummer existiert (Zeilen- und Spaltenzahl ist durch das Feld `belegung` festgelegt) und in der Spalte noch ein Platz frei ist. Die Spaltennummer, die durch den Parameter `spalte` spezifiziert wird, liegt dabei im Intervall $[1, \text{maximale Spaltenzahl}]$ Wenn ja, wird ein Stein des Spielers (der in diesem Fall neu erzeugt werden muss) in der entsprechenden Spalte und dort in der ersten freien Zeile plaziert.

Verwenden Sie dazu jeweils die vorhandenen Informationen (z.B. das Feld `belegung`). Methoden-Parameter, die nicht benötigt werden, dürfen Sie im Methodenkörper ignorieren.

Versehen Sie die Methoden `entfernen` und `bewegen` mit einer einfachen Default-Implementierung (Rückgabewert `false`), da diese hier nicht benötigt werden.

Überschreiben Sie die aus der Oberklasse geerbte Methode `zeichnen`, so dass zunächst die Methode aus der Oberklasse ausgeführt und dann zusätzlich die Spaltennummern darunter geschrieben werden.

(b) Entwickeln Sie eine Klasse `VierGewinntSpiel`, die von der abstrakten Klasse `Spiel` abgeleitet ist und die dieses Spiel realisiert. Hierzu müssen Sie in der Klasse

- einen Konstruktor schreiben, der ein Spiel mit zwei Spielern und einem `VierGewinntBrett` mit sieben Spalten und sechs Zeilen erzeugt sowie
- einen Spielzug eines Spielers implementieren, der so aussieht, das der jeweilige Spieler auf der Kommandozeile augefordert wird, eine Spaltennummer einzugeben, welche dann (mit Hilfe der `IOTools` Klasse) eingelesen wird. Diese Aufforderung soll solange wiederholt werden, bis ein sowohl möglicher als auch korrekter Wert vom Spieler angegeben wurde. Verwenden Sie hierzu die Methode `hinzufuegen` aus der Klasse `Spielbrett`.

Fügen Sie der Klasse eine `main`-Methode hinzu, in der ein "*Vier Gewinnt*" Spiel erzeugt und gestartet wird.

Die während eines Spiels generierte Ausgabe in der Konsole sollte dann so ähnlich wie im folgenden Beispiel ausehen:

```
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
```

```
1 2 3 4 5 6 7

Spieler Nr. 1
Spalte eingeben: 3
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
- - X - - - -

1 2 3 4 5 6 7

Spieler Nr. 2
Spalte eingeben: 4
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
- - X O - - -

1 2 3 4 5 6 7

Spieler Nr. 1
Spalte eingeben: 4
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
- - - X - - -
- - X O - - -

1 2 3 4 5 6 7

Spieler Nr. 2
Spalte eingeben: 5
- - - - - - -
```

```
_ _ _ _ _ _ _
_ _ _ _ _ _ _
_ _ _ _ _ _ _
_ _ _ X _ _ _
_ _ X O O _ _

1 2 3 4 5 6 7

......... (Züge ausgelassen)

Spieler Nr. 1
Spalte eingeben: 6
_ _ _ _ _ _ _
_ _ _ _ _ _ _
_ _ _ O O X _
_ _ _ X X O X
_ _ O X X X O
_ _ X O O X O

1 2 3 4 5 6 7

Spieler Nr. 1 hat gewonnen.
```

Aufgabe 2: Interfaces 7 (=1+2+3+1) Punkte

Das Interface `Number` definiert drei Methoden:

- Die Methode `toDouble` soll den Wert eines Datentyps, der das `Number`-Interface implementiert, in Form eines `double` Werts zurückgeben.
- Die Methode `equals` soll zwei Objekte, die das `Number`-Interface implementieren, auf Gleichheit vergleichen und `true` bzw. `false` zurückgeben, wenn die beiden Objekte in Bezug auf den `double`-Wert, den diese jeweils repräsentieren, gleich sind bzw. ungleich sind.
- Die Methode `compareTo` soll zwei Objekte, die das `Number`-Interface implementieren, vergleichen. Die Methode soll einen ganzzahligen, positiven (negativen) Wert als Ergebnis zurückgeben, wenn das `Number` Objekt, auf dem die Methode aufgerufen wird, größer (kleiner) ist als

das `Number` Objekt, welches als Parameter übergeben wird (jeweils in Bezug auf den `double`-Wert, den die Objekte jeweils repräsentieren). Sind die Werte gleich, soll 0 zurückgegeben werden.

Die Klasse `Bruch` definiert eine Bruchzahl, die durch zwei Attribute für den Zähler und den Nenner repräsentiert wird und die einen Konstruktor zum initialisieren dieser Attribute besitzt. Für eine sinnvolle Ausgabe eines Bruchs ist die Methode `toString` überschrieben.
Die Klasse `TestNumbers` zwei (noch leere) Klassenmethoden, mit denen ein Feld von `Number`-Objekten ausgegeben bzw. sortiert werden sollen.
Darüberhinaus ist eine `main`-Methode definiert, in der ein Feld von $k = 10$ `Bruch`-Objekten angelegt wird. Dieses Feld muss von Ihnen dann noch mit Instanzen der Klasse Bruch gefüllt werden (siehe Aufgabenteil (c)). Anschließend wird der Feld-Inhalt ausgegeben, dann sortiert und nochmals ausgegeben.

(a) Implementieren Sie die Methode `print` in der Klasse `TestNumbers`, in der die String-Repräsentation der Elemente des als Parameter übergebenen Felds `numbers` in einer Zeile ausgegeben werden sollen.

(b) Implementieren Sie die Methode `sort` in der Klasse `TestNumbers`, welche den Inhalt des als Parameter übergebenen Felds aufsteigend sortieren soll. Verwenden Sie hierzu den folgenden Sortier-Algorithmus (den Sie für das Projektblatt P5 für Listen implementiert haben).

Algorithm 1 Bubblesort auf einem Feld

```
1: for i von 1 bis n - 1 (mit n als der Länge des Felds) do
2:     for j von 0 bis n - i do
3:         vergleiche das Element am Index i mit dem am Index (i+1)
4:         if erstes Element ist größer als zweites Element then
5:             tausche Elemente aus
```

(c) Ändern Sie die Klassendefinition der Klasse `Bruch`, so dass diese das Interface `Numbers` implementiert. Implementieren Sie dann die Methoden des Interfaces gemäß der obigen Beschreibung. Hinweis: Um zwei `double`-Werte zu vergleichen, können Sie die Klassenmethode `compare` aus der Klasse `Double` verwenden.

(d) Ergänzen Sie die `main`-Methode der Klasse `TestNumbers` so, dass das Feld `brueche` mit zufälligen Bruchzahlen (Zähler und Nenner sollen hierzu zufällig aus dem Intervall $[1, 20]$ gewählt werden) gefüllt wird.

Die Ausgabe des Programms sollte dann wie in folgendem Beispiel aussehen:

```
(1/7) (1/19) (1/15) (1/9) (1/9) (1/4) (1/16) (1/16) (1/17) (1/13)
(1/19) (1/17) (1/16) (1/16) (1/15) (1/13) (1/9) (1/9) (1/7) (1/4)
```