Abstrakte Klassen, Interfaces und Pakete

Vorbemerkung:

Wir wollen allmählich die Basis für die Entwicklung (einfacher) Spiele in Java schaffen. Dazu erstellen wir in dieser Übung einige Schnittstellen für beliebige Spielobjekte auf einem zweidimensionalen Spielfeld. Dabei werden wir die Interfaces und abstrakte Klassen anwenden.

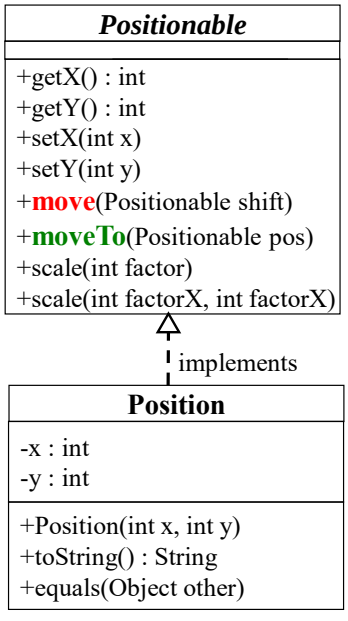
Aufgabe 1: Positionen

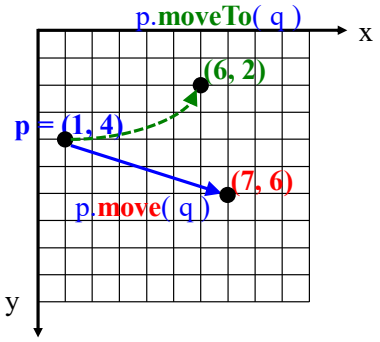
a) Schreiben Sie ein Interface ***Positionable***, welches mindestens die folgenden Methoden deklariert:

- **int getX():** lese die x-Koordinate
- **int getY():** lese die y-Koordinate
- **void move(Positionable shift):** verschiebe die Position um **shift**
- **void moveTo(Positionable pos):** setze die Position auf **pos**
- **void scale(int factor):** skaliere das Objekt (d.h. x- und y-Koordinate) um den Faktor **factor**
- **void scale(int factorX, int factorY):** skaliere das Objekt (d.h. x- und y-Koordinate) um die jeweiligen Faktoren **factorX** bzw. **factorY**

Hinweis: Welche Methoden sind außerdem sinnvoll/erforderlich für Positionen?

b) Implementieren Sie eine Klasse ***Position***, welche das Interface **Positionable** implementiert.
Hinweis: Sie können einfach die Klasse **Point** aus der 2. Übung entsprechend anpassen. Beachten Sie dabei, dass wir jetzt int statt double-Koordinaten verwenden.






```
Position p = new Position(1, 4);
Position q = new Position(6, 2);
p.move( q );        // → p = (1,4) + (6,2) = (7, 6)
p.moveTo( q );      // → p = (6, 2)
```

Aufgabe 2: Pakete und Positionen

Verschieben Sie das Interface **Positionable** in das Package (Paket) **games.basic.position.interfaces** und die Klasse **Position** in das Paket **games.basic.position**.

Hinweis: Beachten Sie, dass Sie dazu auch die Klasse Position anpassen müssen, damit diese auf das Interface **games.basic.position.interfaces.Positionable** zugreifen kann: Entweder Sie verwenden jeweils den vollständigen Interfacenamen oder ein entsprechendes import-Statement.

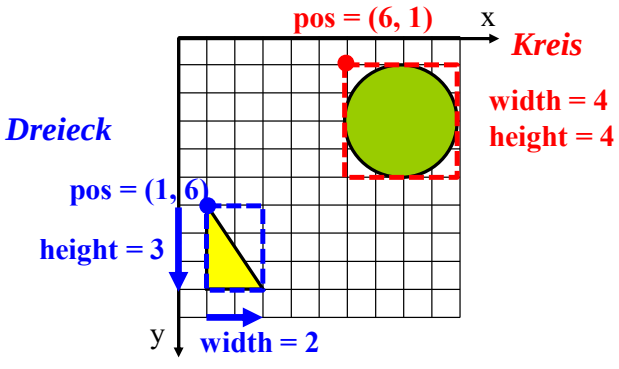
Aufgabe 3: Spielobjekte

Zur Verwaltung und Bewegung von Spielobjekten auf dem Spielfeld betrachten wir zunächst die folgenden Eigenschaften als relevant:

- Die **Position** der linken oberen Ecke des Objektes
- Die maximale **Breite** des Objektes, d.h. dessen Ausdehnung in x-Richtung
- Die maximale **Höhe** des Objektes, d.h. dessen Ausdehnung in y-Richtung

Diese Eigenschaften werden später zum Zeichnen und Bewegen der Spielobjekte benötigt und werden deshalb zunächst als Interface beschrieben.

a) Erstellen Sie ein Interface ***games.basic.gameObjects.interfaces.SimpleGameObject***, d.h. ein Interface ***SimpleGameObject*** im Paket ***games.basic.gameObjects.interfaces***, mit den folgenden Methoden
   - Positionable **getPos**()
   - int **getWidth**()
   - int **getHeight**()



SimpleGameObject
+getPos() : Positionable +getWidth() : int +getHeight() : int

Aufbauend auf diesen abstrakten Methoden können wir Methoden implementieren zum Bewegen von Spielobjekten auf dem Spielfeld sowie zum Vergleichen von deren Positionen, um beispielsweise „Zusammenstöße“ erkennen zu können:

b) Erweitern Sie Sie dazu das Interface **SimpleGameObject** zur Schnittstelle ***games.basic.gameObjects.interfaces.GameObject*** mit den zusätzlichen Methoden[1]

- boolean **isLeftOf**(GameObject other): befindet sich das Objekt (this) links vom anderen Objekt other?
- boolean **isRightOf**(GameObject other)
- boolean **isAbove**(GameObject other)
- boolean **isBelow**(GameObject other)
- boolean **touches**(GameObject other): überprüft, ob sich zwei Spielobjekte gegenseitig berühren, d.h. sich deren Positionen partiell überschneiden (wobei wir nur das umschließende Rechteck jedes Spielobjektes betrachten).

c) Erstellen Sie nun eine (abstrakte) Klasse ***AbstractGameObject*** im Paket ***games.basic.gameObjects***, welche die in **GameObject** deklarierten Methoden mit Hilfe der im Interface **SimpleGameObject** bereitgestellten Methoden implementiert.

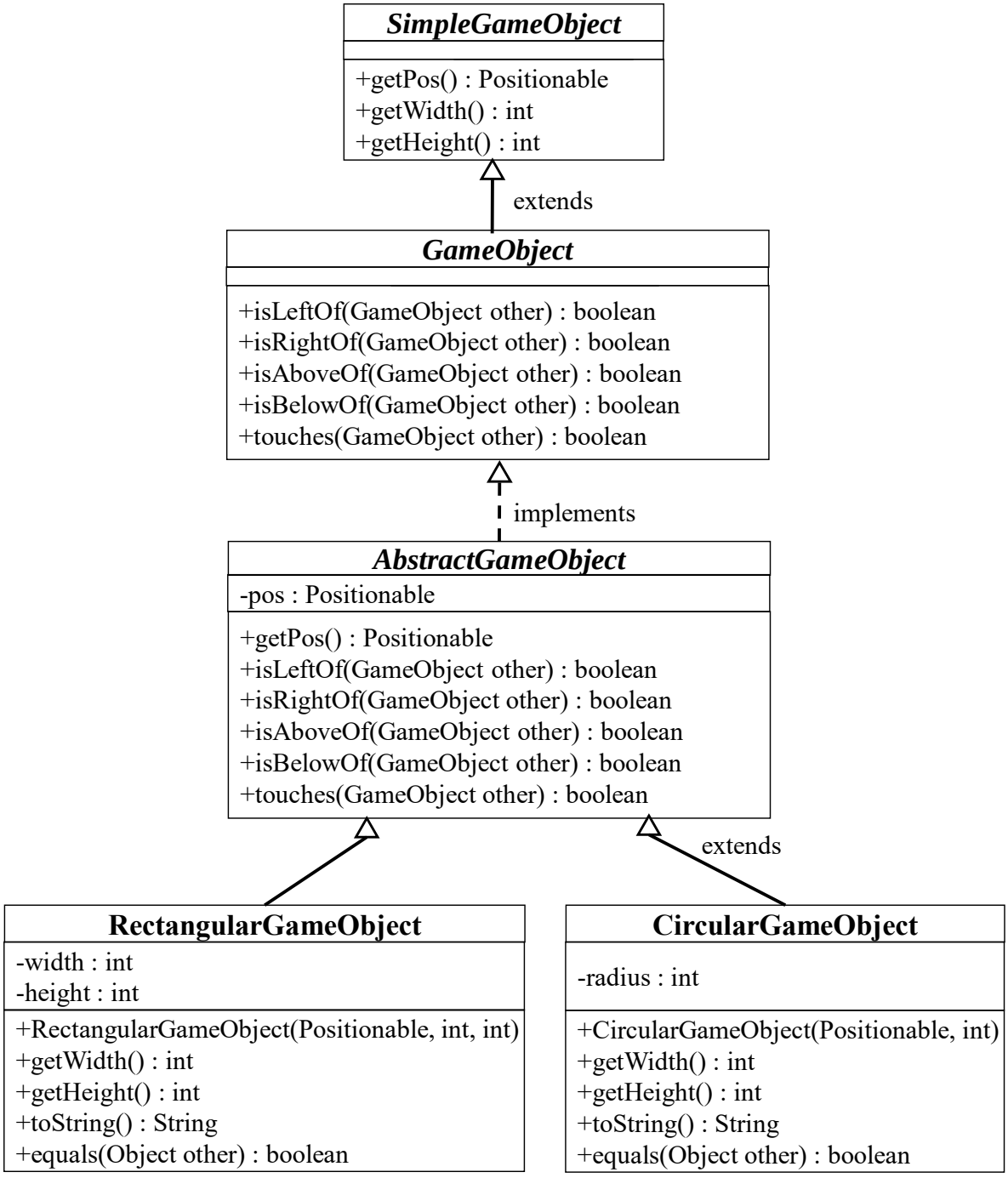
Hinweis: Implementieren Sie zunächst die Methoden **isLeftOf** und **isAbove**. Diese können Sie dann bei der Implementierung der weiteren Methoden verwenden.

d) Leiten Sie davon schließlich konkrete Klassen ab, die ebenfalls im Paket **games.basic.gameObjects** liegen sollen.

- ***RectangularGameObject*** zur Repräsentation rechteckiger Spielobjekte. Diese besitzen intern eine Breite (**width**) und Höhe (**height**).

- ***CircularGameObject*** zur Repräsentation kreisförmiger Spielobjekte. Diese besitzen intern einen Radius (**radius**), und die Breite und Höhe des Objektes entsprechen jeweils dem Kreisdurchmesser.

[1] **Hinweis:** Nachfolgend werden wir nur noch das Interface **GameObject** verwenden. Das Interface **SimpleGameObject** dient hier lediglich zur Veranschaulichung der Vererbung von Interfaces. Wir hätten die Methoden aus **SimpleGameObject** folglich auch direkt im Interface **GameObject** deklarieren können.



e) Schauen Sie noch einmal auf das Interface **SimpleGameObject** (und die implementierenden Klassen **RectangularGameObject** bzw. **CircularGameObject**).
Können Sie etwas über die Position solcher Objekte aussagen?

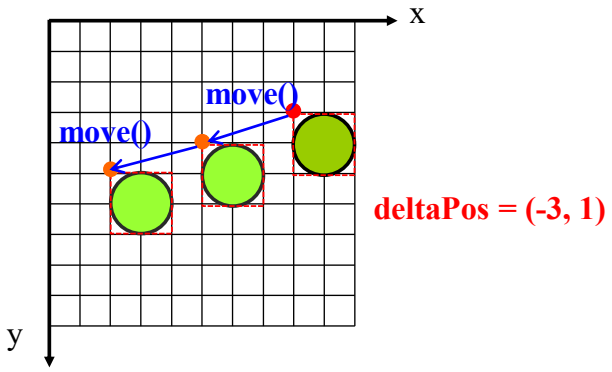
Hinweis: Wie können solche Objekte auf Position (08, 15) gesetzt werden?

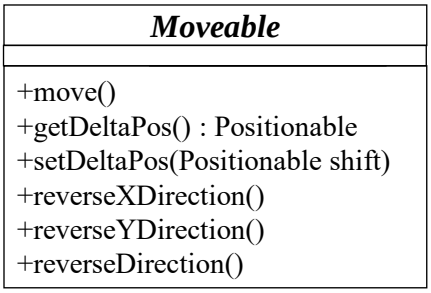
Erweitern Sie das Interface **SimpleGameObject** um dafür notwendige Methoden.

Zusatz-Aufgabe 4: Bewegliche Spielobjekte

Falls Sie noch Zeit und Lust haben, können Sie auch noch die folgende Aufgabe bearbeiten.

Zur Bewegung von Spielobjekten dient die abstrakte Methode **move().** Diese soll das Objekt gemäß eines Verschiebevektors **deltaPos** in x- und y-Richtung verschieben, d.h. dessen Position entsprechend anpassen.





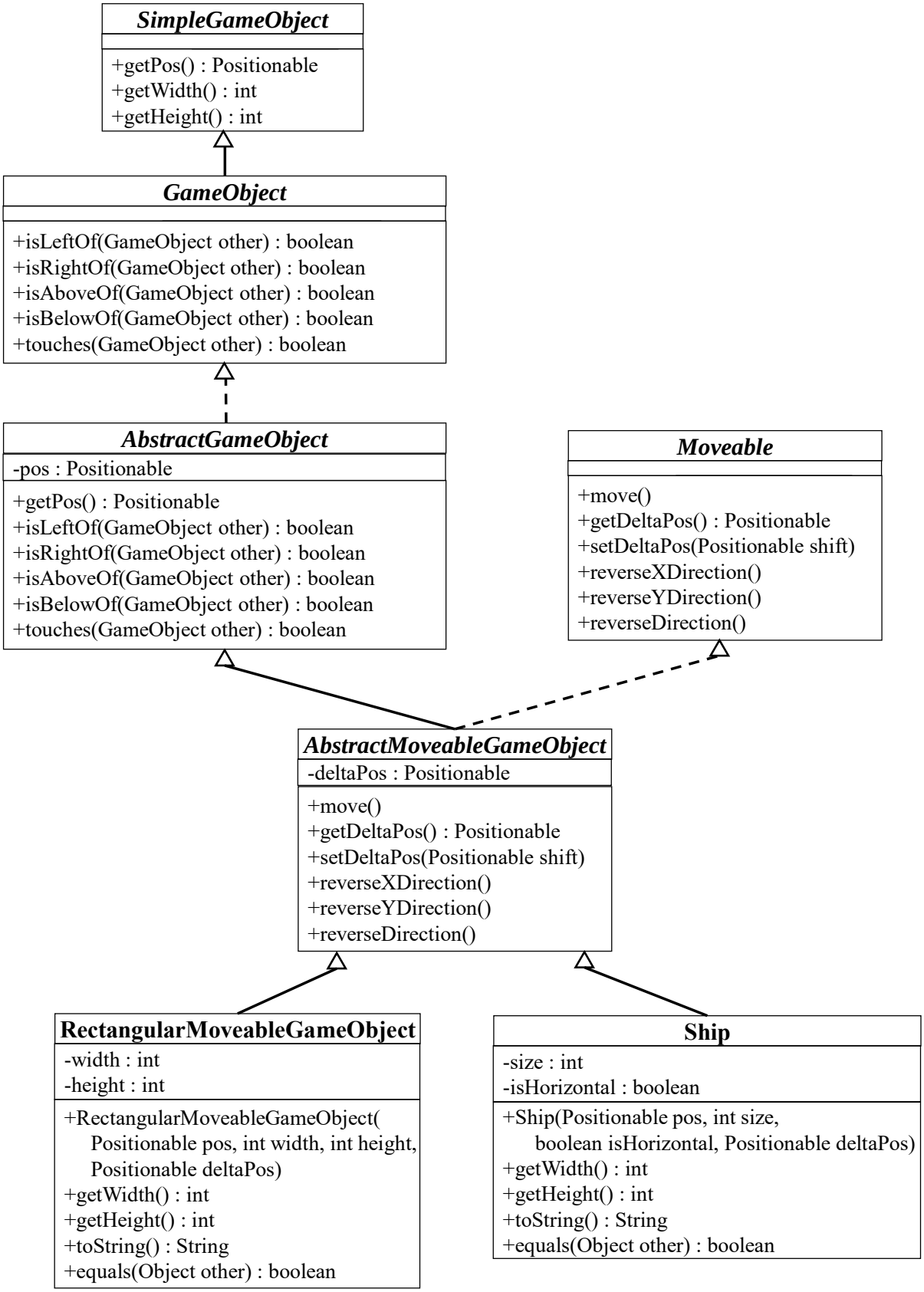
a) Erstellen Sie ein Interface **games.basic.gameObjects.interfaces.*Moveable*** mit den folgenden Methoden
   - void **move**()
   - void **setDeltaPos**(Positionable shift)
   - Positionable **getDeltaPos**()
   - void **reverseXDirection**(): Ändert die Bewegungsrichtung
   - void **reverseYDirection**()
   - void **reverseDirection**()

b) Implementieren Sie diese Methoden in einer (abstrakten) Klasse **games.basic.gameObjects.moveable.*AbstractMoveableGameObject***, welche Sie von **AbstractGameObject** ableiten können. Die Klasse soll

c) Erstellen Sie schließlich (im Paket **games.basic.gameObjects.moveable*)*** zwei konkrete Klassen

   - ***RectangularMoveableGameObject*** für rechteckige Spielobjekte, und

   - ***Ship*** für eindimensionale Spielobjekte, d.h. Objekte mit Höhe = 1 und Breite = size (bei horizontaler Ausrichtung des Schiffes), bzw. Höhe = size und Breite = 1 (bei vertikaler Ausrichtung).


SimpleGameObject
+getPos() : Positionable
+getWidth() : int
+getHeight() : int
GameObject
+isLeftOf(GameObject other) : boolean
+isRightOf(GameObject other) : boolean
+isAboveOf(GameObject other) : boolean
+isBelowOf(GameObject other) : boolean
+touches(GameObject other) : boolean
AbstractGameObject
-pos : Positionable
+getPos() : Positionable
+isLeftOf(GameObject other) : boolean
+isRightOf(GameObject other) : boolean
+isAboveOf(GameObject other) : boolean
+isBelowOf(GameObject other) : boolean
+touches(GameObject other) : boolean
Moveable
+move()
+getDeltaPos() : Positionable
+setDeltaPos(Positionable shift)
+reverseXDirection()
+reverseYDirection()
+reverseDirection()
AbstractMoveableGameObject
-deltaPos : Positionable
+move()
+getDeltaPos() : Positionable
+setDeltaPos(Positionable shift)
+reverseXDirection()
+reverseYDirection()
+reverseDirection()
RectangularMoveableGameObject
-width : int
-height : int
+RectangularMoveableGameObject(
Positionable pos, int width, int height,
Positionable deltaPos)
+getWidth() : int
+getHeight() : int
+toString() : String
+equals(Object other) : boolean
Ship
-size : int
-isHorizontal : boolean
+Ship(Positionable pos, int size,
boolean isHorizontal, Positionable deltaPos)
+getWidth() : int
+getHeight() : int
+toString() : String
+equals(Object other) : boolean

Zusatz-Aufgabe 5: Intelligenter Kühlschrank

c) Schreiben Sie ein Interface ***Validity***, welches mindestens die folgenden Methoden deklariert:
- **boolean hasExpired():** Hat das Lebensmittel sein Haltbarkeitsdatum überschritten?
- **int validity():** Haltbarkeit in Tagen (evtl. negativ!)

b) Implementieren Sie eine Klasse ***AbstractFood***, welche das Interface **Validity** implementiert.

Hinweis: Dort soll die Methode **hasExpired()** in Abhängigkeit von der Haltbarkeit (**validity()**) implementiert werden.

c) Erstellen Sie eine Klasse ***Food***, welche von **AbstractFood** abgeleitet ist.

Hinweis: Denken Sie an die Definition
- der benötigten Attribute (u.a. wird das Mindesthaltbarkeitsdatum **bestBefore** benötigt)
- und eines geeigneten Konstruktors.

d) Verschieben Sie das Interface **Validity** in das Package (Paket) **fridge.food.interfaces** und die Klasse **AbstractFood** in das Paket **fridge.food**.

Hinweis: Beachten Sie, dass Sie dazu auch die Klasse AbstractFood anpassen müssen, damit diese auf das Interface **fridge.food.interfaces.Validity** zugreifen kann: Entweder Sie verwenden jeweils den vollständigen Interfacenamen oder ein entsprechendes import-Statement.

e) Verschieben Sie analog die anderen Klassen in das Paket **fridge.food**.

Zusatz-Aufgabe 6 (für Java-Experten): Java-Rätsel – Strings/Zahlen

a) Rätsel (StringBuffer): Das folgende Programm gibt ein Wort aus unter Verwendung eines Zufallszahlengenerators, um das erste Zeichen auszuwählen. Beschreiben Sie das Verhalten des Programms:

```
import java.util.Random;

public class Reim {
      private static Random rnd = new Random();

      public static void main(String[] args) {
            StringBuffer word = null;
            switch (rnd.nextInt(2)) {
                  case 1:  word = new StringBuffer('P');
                  case 2:  word = new StringBuffer('G');
                  default: word = new StringBuffer('M');
            }
            word.append('a');
            word.append('i');
            word.append('n');
            System.out.println(word);
      }
}
```

b) Rätsel (ungerade Zahlen): Die folgende Methode versucht zu ermitteln, ob die übergebene Zahl ungerade ist. Funktioniert diese Methode?

```
      public static boolean isOdd(int i) {
            return i % 2 == 1;
      }
```

c) Rätsel (Hexadezimal-Zahlen): Das folgende Programm addiert zwei Hexadezimal-Zahlen und gibt das Ergebnis hexadezimal aus. Was gibt das Programm aus?

```
public class JoyOfHex {
      public static void main(String[] args) {
            System.out.println(
                  Long.toHexString(0x100000000L + 0xcafebabe));
      }
}
```