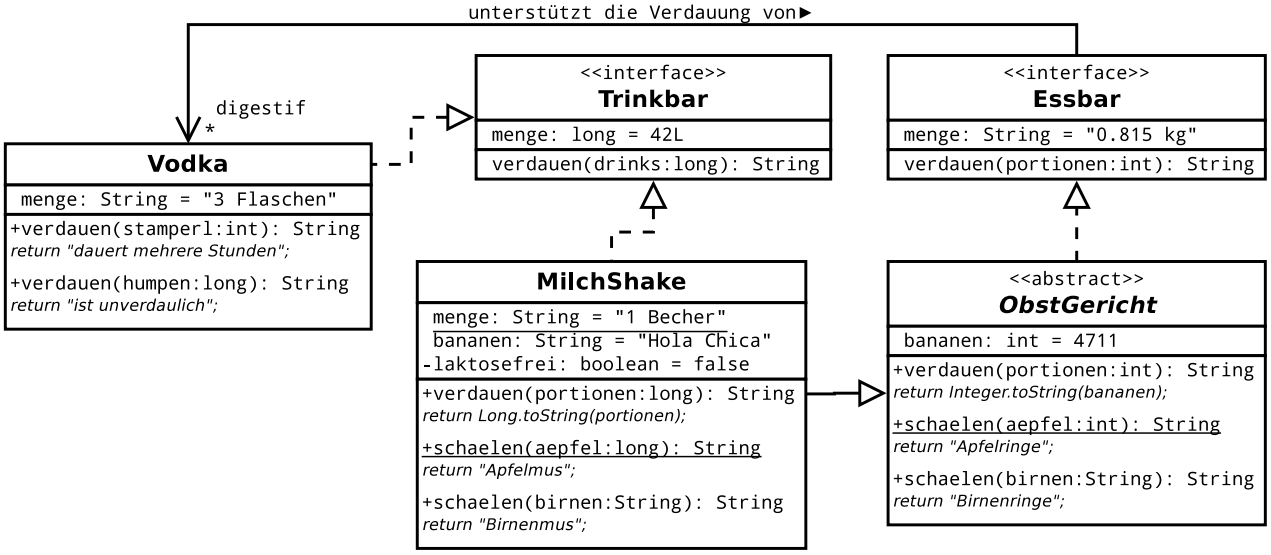
FAU, Informatik 2, AUD-Team
aud@i2.cs.fau.de

7. Übung

Abgabe bis 12.12.2016, 10:00 Uhr

Einzelaufgabe 7.1: OOP/Polymorphie - Nachtisch — 10 EP

Gegeben ist das folgende UML-Klassendiagramm – die Rückgabewerte der Methoden sind dabei direkt unter den Methoden als Kommentare vermerkt:



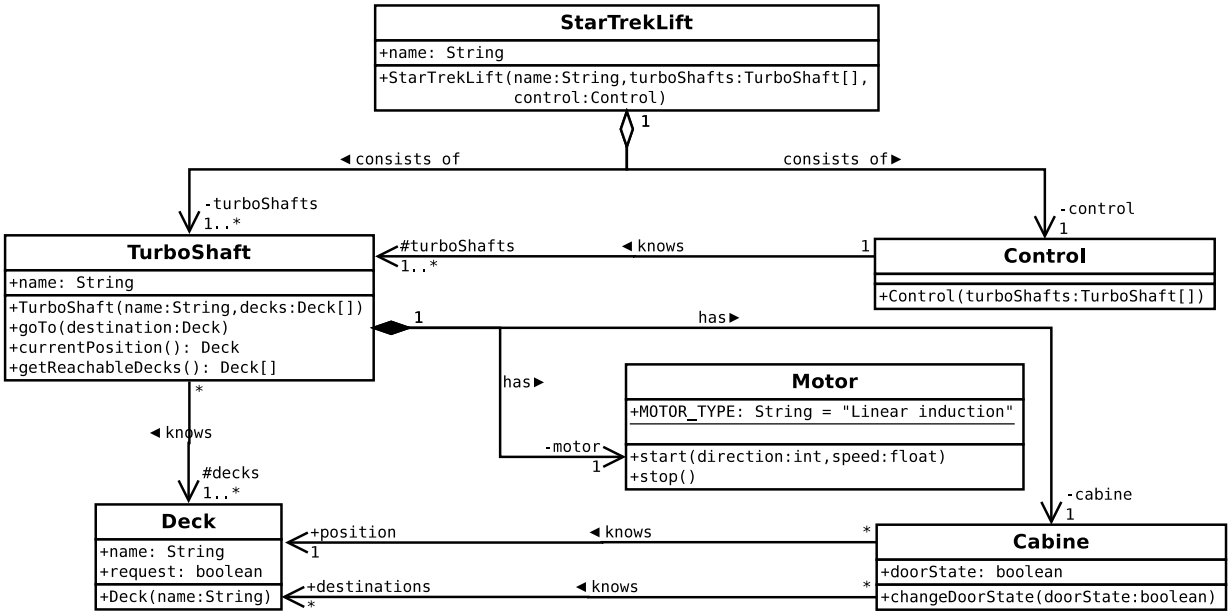
Geben Sie an, welche Ausgaben die mit (1)–(10) gekennzeichneten Zeilen im folgenden Java-Programm erzeugen und begründen Sie Ihre Antwort kurz.

```
public class NachtischTest {
    public static void main(String[] args) {
        Trinkbar tm = new MilchShake();

        System.out.println(tm.menge);            // *** (1) ***

        Essbar em = new MilchShake();

        System.out.println(em.verdauen(3));      // *** (2) ***
        System.out.println(em.menge);            // *** (3) ***

        ObstGericht om = new MilchShake();

        System.out.println(om.verdauen(4));      // *** (4) ***
        System.out.println(om.menge);            // *** (5) ***

        System.out.println(om.schaelen(          // *** (6) ***
                Integer.MAX_VALUE + 123));

        System.out.println(om.schaelen("Kiwi")); // *** (7) ***

        MilchShake mm = new MilchShake();

        System.out.println(mm.menge);            // *** (8) ***

        Trinkbar tv = new Vodka();

        System.out.println(tv.verdauen(1));      // *** (9) ***

        System.out.println(tm.verdauen(2));      // *** (10) ***
    }
}
```

Geben Sie Ihre Lösung als `Nachtisch.pdf` im *EST* ab.

Einzelaufgabe 7.2: StarTrekLift — 21 EP

Gegeben ist das folgende UML-Klassendiagramm eines Liftsystems der Serie Star Trek (vgl. hier), welches z.B. in der USS Enterprise verbaut ist.



Setzen Sie das Diagramm in *übersetzbaren* Java-Code um. Erstellen Sie hierzu eine Datei `StarTrekLift.java` und fügen Sie alle im Diagramm gezeigten Bestandteile dort ein. Geben Sie Ihre Lösung (*übersetzbare* `StarTrekLift.java`) im *EST* ab.

Wichtig: Verschachteln Sie keine Klassen oder Schnittstellen ineinander. In Java können mehrere Klassen und Schnittstellen in einer Datei nebeneinander stehen, sofern man bei allen Klassen/Schnittstellen (die nicht dem Dateinamen entsprechen) jeweils das Schlüsselwort `public` weglässt.

Bei Konstruktoren und Methoden muss lediglich der Methodenrahmen implementiert werden, der die im UML-Diagramm gezeigte jeweilige Signatur umsetzt. Methoden- und Konstruktorenkörper können leer gelassen werden, müssen aber *übersetzbar* sein, d.h. ggf. muss bei Methoden eine gültige `return`-Anweisung eingefügt werden. Für „unbegrenzt große“ Reihungen verwenden Sie in Java jeweils Arrays.

Gruppenaufgabe 7.3: Objektorientierung: AwesomMMORPG — 29 GP

Sie haben ein wenig freie Zeit. Daher beschließen Sie zusammen mit einem Kommilitonen ein MMORPG zu implementieren. Jeder Spieler steuert einen Avatar `AwesomAvatar`. Dieser hat `lifePoints`, `strength`, `intelligence`, `mana` und `experience`. Die maximalen `lifePoints` werden aus dem Produkt von `experience` und `strength` berechnet. Das maximal verfügbare `mana` aus dem Produkt von `experience` und `intelligence`. Wenn ein Spieler einen Avatar erstellt, so beginnt dieser das Spiel mit der für ihn maximalen Anzahl an `lifePoints` und `mana`. Jeder Avatar erscheint im Spiel zudem mit einer Waffe `AwesomWeapon`.
In dieser Aufgabe werden Sie zahlreiche Setter- und Getter-Methoden implementieren. Zugriffe auf Instanz-Variablen müssen *ausschließlich* über diese Methoden erfolgen.
Laden Sie die Dateien AwesomAvatar.java, AwesomWeapon.java, AwesomSpell.java und AwesomAvatarPublicTest.java von der AuD-Homepage.

a) Bearbeiten Sie die Klasse `AwesomAvatar`:

Ergänzen Sie den Konstruktor `AwesomAvatar(int strength, int intelligence, int experience, AwesomWeapon weapon)`. Die Bedeutung der übergebenen Parameter wird aus der Beschreibung oben ersichtlich.

Implementieren Sie die Getter-Methoden `getLifePoints()`, `getMaxLifePoints()`, `getStrength`, `getIntelligence()`, `getMana()`, `getMaxMana()` und `getExperience()`. Diese geben die jeweiligen Werte zurück.

Implementieren Sie die Setter-Methoden `setLifePoints()`, `addLifePoints()`, `removeLifePoints()`. Diese setzen die Lebenspunkte auf einen bestimmten Wert, fügen Lebenspunkte hinzu oder entfernen sie, wobei folgende Bedingungen eingehalten werden müssen: Lebenspunkte können nicht über ihren Maximalwert erhöht werden, negative Lebenspunkte hingegen sind möglich. `addLifePoints()` soll keine Lebenspunkte entfernen und `removeLifePoints()` keine hinzufügen können. Die Methoden sollen nur bei positiven Übergabewerten Änderungen vornehmen und andernfalls nichts tun. Implementieren Sie die Setter-Methode `setMana(int mana)` äquivalent zu `setLifePoints()`.

Implementieren Sie die Methoden `setExperience()`, `setStrength()` und `setIntelligence`. Diese sollen die genannten Werte auf die übergebenen Werte setzen. Hierbei sind positive wie negative Werte erlaubt.

Implementieren Sie die Methoden `getWeapon()` und `setWeapon(Weapon weapon)`.

Implementieren Sie die Methode `rest()`. Diese setzt `mana` und `lifePoints` zurück auf das Maximum.

b) Bearbeiten Sie die Klasse `AwesomWeapon`:

Implementieren Sie den Konstruktor `AwesomWeapon(int damage, int wear)`. Die Parameter `damage` und `wear` geben den Schaden an, der von der Waffe ausgeht sowie die Anzahl der Attacken, die ausgeführt werden können bis die Waffe nicht mehr funktioniert.

Implementieren Sie die Getter-Methoden `getDamage()` und `getWear()`. Diese sollen den Schaden, den die Waffe zufügt bzw. die aktuelle *Haltbarkeit* `wear` zurückgeben. Implementieren Sie zudem die Setter-Methoden `setWear(int wear)` und `setDamage(int damage)`. Diese setzen Haltbarkeit und Schaden der Waffe auf die übergebenen Werte.

Implementieren Sie die Methode `attack()`. Diese prüft zunächst, ob `wear` größer 0 ist. Ist die Waffe aufgebraucht, so soll 0 zurückgegeben werden. Falls die Waffe noch funktioniert soll `wear` um 1 reduziert werden und der Schaden, den die Waffe zufügt, zurückgegeben werden.

c) Bearbeiten Sie die Klasse `AwesomAvatar` weiter:

Bearbeiten Sie die Methode `attack(AwesomAvatar avatar)`. Diese führt eine Attacke mit der Waffe des Spielers auf den übergebenen Avatar aus. Sollten nach der Attacke die Lebenspunkte des Angegriffenen auf 0 oder darunter liegen, so werden dessen Erfahrungspunkte zu den Erfahrungspunkten des Angreifers hinzuaddiert und falls die Waffe des Gegners sowohl mehr Schaden anrichten kann als auch weniger abgenutzt ist, wird die eigene Waffe durch diese ausgetauscht und die Leiche erhält die schwächere Waffe (Leichenfleddern mit Zurücklegen).

Immer wenn ein anderer Spieler besiegt wurde, bekommt der eigene Avatar je einen Punkt auf `strength` und `intelligence`. Implementieren Sie hierfür die Methode `levelUp()`. Erweitern Sie die Methode `attack(AwesomAvatar avatar)` so, dass `levelUp()` ausgeführt wird, wenn der Gegner tot ist.

d) Bearbeiten Sie die Klasse `AwesomSpell`. Implementieren Sie den Konstruktor `AwesomSpell(int damage)`. Implementieren Sie die Getter-Methoden `getDamage()` und `getCost()`. Die Kosten eines Zauberspruchs belaufen sich auf das Zehnfache des Schadens. Implementieren Sie die Setter-Methode `setDamage(int damage)`.

Bearbeiten Sie die Klasse `AwesomAvatar` zum letzten Mal. Implementieren Sie die Methode `castSpell(AwesomAvatar avatar, AwesomSpell spell)`. Ein Zauberspruch kostet Mana. Ist nicht genügend Mana vorhanden, so wird der Zauberspruch nicht ausgeführt bzw. abgebrochen. Wenn genügend Mana vorhanden ist, wird die benötigte Menge Mana von der vorhandenen subtrahiert.

Zaubersprüche treffen die Waffe des Gegners und reduzieren deren Haltbarkeit um den vom Zauberspruch angerichteten Schaden.

Geben Sie Ihre Lösung als `AwesomAvatar.java`, `AwesomWeapon.java` und `AwesomSpell.java` über EST ab.

31 EP + 29 GP = 60 Punkte