Aufgabenzettel 4
Vererbung und Polymorphie

Dieser Aufgabenzettel muss bis zum **25.5., 12:00 Uhr**, elektronisch in Moodle als Gruppe von 2-3 Studierenden abgeben werden. Spätere Abgaben sind nicht möglich.

Die Abgabe erfolgt als ZIP-Datei, die folgende Konventionen einhalten **muss**:

- Dateiname: p2-assignment-4-group-*Gruppennummer*.zip
  (Beispiel: Gruppe 15 gibt eine Datei mit dem Namen p2-assignment-4-group-15.zip ab)
- Die folgenden Dateien müssen sich in der Abgabe befinden (Ordner beachten!):
  - `src/de/hrw/progra2/assignment4/AbstractBeing.java`
  - `src/de/hrw/progra2/assignment4/AbstractGameElement.java`
  - `src/de/hrw/progra2/assignment4/AbstractHumanParticipant.java`
  - `src/de/hrw/progra2/assignment4/AbstractThing.java`
  - `src/de/hrw/progra2/assignment4/CanAttack.java`
  - `src/de/hrw/progra2/assignment4/CanBeCarriedInInventory.java`
  - `src/de/hrw/progra2/assignment4/CanBeEaten.java`
  - `src/de/hrw/progra2/assignment4/Fruit.java`
  - `src/de/hrw/progra2/assignment4/GameAdmin.java`
  - `src/de/hrw/progra2/assignment4/GameUtil.java`
  - `src/de/hrw/progra2/assignment4/MarzipanWeapon.java`
  - `src/de/hrw/progra2/assignment4/Monster.java`
  - `src/de/hrw/progra2/assignment4/PayingCustomerPlayer.java`
  - `src/de/hrw/progra2/assignment4/Stone.java`
  - `src/de/hrw/progra2/assignment4/Vector3D.java`
  - `src/de/hrw/progra2/assignment4/Weapon.java`
  - `src/de/hrw/progra2/assignment4/World.java`
- Die folgende PDF-Datei muss sich im Wurzelverzeichnis der Abgabe befinden:
  - `class-diagram.pdf`
- Falls Methodenköpfe (Rückgabetyp, Methodennamen, Parameter) vorgegeben sind, dürfen sie nicht geändert werden. Wenn Sie eine Methode nicht bearbeiten wollen, stellen Sie trotzdem sicher, dass ein gültiger Wert (z.B. null) zurückgegeben wird, damit der Code kompiliert.
- Alle Java-Dateien müssen in UTF-8 kodiert sein und sich im Ordner `src/de/hrw/progra2/assignment4/` befinden und kompilieren.
- Die ZIP-Datei darf keine Dateien mit Endung `.class` oder `.jar` enthalten.

Abgaben, die diese Konventionen nicht einhalten, werden mit 0 Punkten bewertet.
Tipp: Verwenden Sie die Vorlage p2-assignment-4-template.zip. Benutzen Sie das Online-Tool unter https://codingprof.hs-rw.de/csf/?module=p2&assignment=4, um Ihre Datei vor der Abgabe zu prüfen.

Zur Abnahme des Übungszettels müssen Sie mindestens 50% der Punkte erreichen. Beachten Sie bitte, dass Plagiate und Internetlösungen zu der Wertung „nicht bestanden“ im Praktikum führen. Entwickeln Sie im eigenen Interesse selbstständig eine Lösung ohne Hilfe durch Dritte. Fachliche Fragen können Sie gerne im Diskussionsforum stellen.

Aufgabe: Implementierung einer Klassenhierarchie (27 Punkte)

Gegeben ist die folgende Klassenhierarchie, die die Elemente in einer virtuellen Spielwelt modelliert:

- Die abstrakte Klasse **AbstractGameElement** bildet die abstrakte Oberklasse aller Spielelemente. Sie besitzt als Attribute einen Namen `name` (String) und eine Position in drei dimensionalen Welt `position` (Vector3D).

  Hinweis: Die Klasse Vector3D ist vorgegeben und muss nicht implementiert werden.

- Die abstrakte Klasse **AbstractThing** ist ein Gegenstand in der Welt, von AbstractGameElement abgeleitet und verfügt über folgendes Attribut:

  - `isVisible` gibt an, ob der Gegenstand sichtbar ist

- Die Schnittstelle **CanBeEaten** wird von Objekten implementiert, die gegessen werden können. Die Schnittstelle schreibt eine parameterlose Methode *getEaten* vor, die die Anzahl der Lebensenergiepunkte als long zurückgibt, die durch den Verzehr des Objekts erworben werden. Kann ein essbares Objekt gerade nicht verzehrt werden, z.B. weil es aktuell nicht sichtbar ist, wird 0 zurückgegeben.

- **CanBeCarriedInInventory** ist eine Markierungsschnittstelle, die angibt, dass ein Spielelement in einem Inventar (Rucksack) getragen werden kann.

- Die abstrakte Klasse **AbstractBeing** ist ein Lebewesen in der Welt, von AbstractGameElement abgeleitet und verfügt über folgende Attribute:

  - `maxEnergy` ist ein long-Wert, der die maximale Lebensenergie eines Lebewesens darstellt

  - `currentEnergy` ist ein long-Wert, der die aktuelle Lebensenergie eines Lebewesens darstellt

  Außerdem besitzt die Klasse mehrere Methoden:

    - **isLiving()** gibt genau dann true zurück, wenn die aktuelle Lebensenergie größer ist als 0.
    - **revive()** setzt die aktuelle Lebensenergie auf die maximale Lebensenergie.
    - **addEnergy(*energy* : long)** fügt die aktuellen Lebensenergie den Betrag *energy* hinzu. Die maximale Lebensenergie kann nicht überschritten werden.
    - **deductEnergy(*energy* : long)** zieht den Betrag *energy* von der aktuellen Lebensenergie ab. Die Wert 0 kann dabei nicht unterschritten werden.
    - **eat(*object* : CanBeEaten)** ist ein Objekt, das CanBeEaten implementiert und fügt die zurückgegeben Lebensenergie der eigenen Lebensenergie hinzu.

- Die Schnittstelle **CanAttack** wird von Objekten implementiert, die Lebewesen attackieren können. Die Schnittstelle schreibt eine Methode *attack* vor, die als Parameter ein Lebewesen (AbstractBeing) erhält und keinen Wert zurückgibt.
- Die Klasse **Fruit** stellt eine Frucht dar und erbt von **AbstractThing**. Sie besitzt als Attribut den long-Wert `calories`, der die Anzahl der Kalorien der Frucht repräsentiert. Eine Frucht kann gegessen werden, wenn sie sichtbar ist. Sie gibt dann ihre Kalorienzahl als Lebensenergie zurück und wird unsichtbar. Eine Frucht kann in ein Inventar aufgenommen werden.
- Die Klasse **Stone** stellt einen Stein dar und erbt von **AbstractThing**. Sie besitzt als Attribut den double-Wert `weight`, der das Gewicht des Steins in Kilo repräsentiert. Eine Stein kann in ein Inventar aufgenommen werden.
- Die Klasse **Weapon** stellt eine Waffe dar und erbt von **AbstractThing**. Sie besitzt als Attribut den double-Wert `cost`, der die Kosten der Waffe in Euro repräsentiert. Eine Waffe kann in ein Inventar aufgenommen werden.
- Die Klasse **MarzipanWeapon** ist eine spezielle Waffe. Da sie aus Marzipan besteht, kann man sie essen. Für das Aufessen einer sichtbaren Waffe erhält man 50 Lebenspunkte. Die Waffe wird nach dem Essen in der Spielwelt unsichtbar.
- Ein Monster wird durch die Klasse **Monster** repräsentiert, die von **AbstractBeing** erbt. Ein Monster hat ein long-Attribut `strength`, das dessen Stärke darstellt. Ein Monster kann andere Lebewesen angreifen. Dabei wird den angegriffenen Lebenswegen `strength` Lebenspunkte abgezogen.
- Die abstrakte Klasse **AbstractHumanParticipant** erbt von **AbstractBeing** und stellt einen menschlichen Teilnehmer in der virtuellen Welt dar. Er besitzt das Attribut `loginName`, das einen Login-Namen bezeichnet.

- Die Klasse **PayingCustomerPlayer** erbt von **AbstractHumanParticipant** und stellt einen zahlenden, menschlichen Spieler da. Als Attribute besitzt er den double-Wert `amountSpentPerMonth`, der den monatlichen Betrag angibt, den die Person für das Spiel gibt. Die Klasse enthält außerdem ein Attribut `inventoryItems`. Es stellt ein Inventar dar, das durch ein Array (oder eine Liste) dargestellt wird, dessen Elemente CanBeCarriedInInventory implementieren.
  Die Klasse **PayingCustomerPlayer** erweitert die Implementierung von deductEnergy dahingehend, das ein Spieler nach seinem virtuellen Versterben an das Zentrum der Welt (Position 0,0,0) zurückgesetzt und wieder zum Leben erweckt wird.

- Die Klasse **GameAdmin** erbt von **AbstractHumanParticipant** und stellt ein Spiel-Administrator dar. Dabei handelt es sich um einen menschlichen Teilnehmer, der in der Welt nach dem rechten sieht. Die Klasse besitzt ein Attribut `visibileForOtherPlayers`, das angibt, ob der Spieler in der Welt sichtbar ist oder nicht.
  Ein Spiel-Administrator hat immer 100 Energiepunkte und kann diese nicht verlieren. Die Klasse überschreibt daher die Methode deductEnergy mit einer leeren Implementierung.

- Die Klasse **World** repräsentiert die Spielwelt. Sie enthält als Attribut eine Liste von Spielelementen. Sie verfügt über eine Methode um ein Element der Spielwelt hinzuzufügen.

Die Klasse **GameUtil** bietet die folgenden **statischen** Hilfsmethoden:

- Die Methode **eatStuff** erhält als Parameter ein Lebewesen und eine Liste von Objekten. Die Methode sorgt dafür, dass das Lebewesen alle essbaren Objekte isst.

- Die Methode **getPotentialInventoryItems** erhält eine Referenz auf eine Welt als Parameter und gibt ein Array von Objekten zurück, die sich in einem Inventar ablegen lassen.

- Die Methode **getMonthlyProfit** erhält eine Referenz auf eine Welt als Parameter und gibt den Gewinn zurück, den die Welt im Monat erzeugt. Jedes Element in der Welt kostet 1 €. Dafür generieren zahlende, menschliche Spieler (PayingCustomerPlayer) Umsatz.

Hinweis: Die Klasse GameUtil befindet sich in der Vorlage und muss nur ausgefüllt werden.

Ihre Aufgaben sind die folgenden:

1. Erstellen Sie ein UML-Diagramm, das die folgenden Klassen **mit den Schnittstellen**, die sie implementieren, darstellt: **AbstractGameElement, AbstractThing, Fruit, Stone, Weapon, MarzipanWeapon**. (8 Punkte)

   Das Klassendiagramm muss sich als PDF-Dokument `class-diagram.pdf` im Wurzelverzeichnis Ihrer Abgabe befinden.

2. Implementieren Sie die Klassenhierarchie in Java. (17 Punkte)

   - Achten Sie dabei auf sämtliche objektorientierten Konzepte (Information Hiding etc.).
   - JedeKlasse sollte angemessene Konstruktoren, Setter/Getter und eine toString-Methode besitzen. Jede toString-Methode sollte alle Attribute ausgegeben.
   - Überlegen Sie, welche Klassen welche Schnittstellen implementieren.

3. Schreiben Sie ein Hauptprogramm, das alle Objekte und Hilfsmethoden angemessen demonstriert. (2 Punkte)