Aufgabe 4.6: Battle Arena

Deine Rolle
Du bist Java Applikationsentwickler bei DragenFire und dort für die Entwicklung von Softwarearchitekturen für Java-Prototypen von neue Computerspielen zuständig.

Die Situation
In der Firma wird daran gedacht, eine neues Computerspiel zu entwickeln, das den runden-basierten PvP-Kampf in einer Arena zum Thema hat. Dabei sollen die unterschiedlichsten Monstertypen (im Stil von Pokemon) zum Einsatz kommen.

Dein Ziel
Du sollst einen Java-Prototyp für ein neues Spiel „Battle Arena“ entwickeln. In einer Battle Arena treten pro Kampf zwei Rivalen gegeneinander an. Es handelt sich dabei um einen Java-Kommandozeilen-Prototyp, der einige Basisfunktionalitäten einer rundenbasierten PvP-Kampfarena bereitstellt (Details siehe unten) und möglichst gut mit neuen Charakterklassen erweiterbar ist, Code gut wiederverwendet bzw. die Wiederverwendung von Code gut möglich macht (d.h. z.B. keine Codeduplikate aufweist), insgesamt gut wartbar ist und damit als Beispiel für eine mögliche Architektur für ein entsprechendes Spiel dienen kann.

Das erwartete Produkt

Aufbau der Basisklasse
Alle Kämpfer-Charatere haben folgenden Aufbau:

- **Name** (wird bei der Konstruktion definiert)
- **Lebenspunkte** (0 bis 100, Start mit 100 Lebenspunkten)
- **specialAbilityActive** (bool’sche Variable, die angibt, ob die Spezialfähigkeit aktiviert wurde)
- Getter und Setter für die Datenfelder (entsprechend den Vorgaben unten)
- **public void getDamage(int points)**: wird aufgerufen, wenn ein Charakter angegriffen wurde und ihm dadurch Lebenspunkte abgezogen werden.
- **public void attack(Charakter enemy)**: wird aufgerufen, wenn ein Charakter einen anderen Charakter angreift.

Je nach konkreter Charakterklasse der Kämpfer gelten folgende spezifische Eigenschaften (es sind bisher nur zwei Charakterklassen vorgesehen, die später jedoch beliebig erweitert werden):

- **Drachen**
  - können **angreifen**
    - der Angriffswert der Drachen liegt zwischen 20 und 25 Punkten (zufällig bei jedem Angriff gewählt)
    - der errechnete Angriffswert wird bei einem Angriff dem Gegner bei den Lebenspunkten abgezogen.
  - können eine **Spezialfertigkeit „fly“ aktivieren**
    - Malus auf den Basis-Angriffswert bei aktiver Spezialfertigkeit: zwischen 5 und 10 Punkte (zufällig) Abzug auf den Basiswert. Bei aktivierter Spezialfertigkeit ist ein Drachenangriff demnach nicht so effektiv.
    - Bonus auf die Lebenspunkte bei aktiver Spezialfertigkeit: 10 Lebenspunkte zusätzlich bei aktivierter Spezialfähigkeit. Der Drache ist in der Luft kraftvoller.
  - können die **Spezialfertigkeit wieder deaktivieren**
    - Bonus und Malus werden wieder aufgehoben
  - können **Schaden** nehmen (bei einem Angriff)

- **Zwerge**
  - können **angreifen**
    - der Angriffswert der Zwerge liegt zwischen 15 und 25 Punkten (zufällig bei jedem Angriff gewählt)
    - der errechnete Angriffswert wird bei einem Angriff dem Gegner bei den Lebenspunkten abgezogen.
  - können eine **Spezialfertigkeit „Zwergenkopfnuss“ aktivieren**
    - Die Spezialfähigkeit lässt sich nur aktivieren, wenn der Zwerg weniger als 50 Lebenspunkte hat.
    - Bonus auf den Angriffswert: Ist die Spezialfähigkeit aktiviert und ist sie erfolgreich (Wahrscheinlichkeiten siehe unten), dann wird der Basis-Angriffswert des Zwerges für den aktuellen Angriff verdoppelt.
    - Malus auf den Angriffswert: Ist die Spezialfähigkeit aktiviert und ist sie nicht erfolgreich (Wahrscheinlichkeiten siehe unten), dann wird der Basis-Angriffswert des Zwerges für den aktuellen Angriff halbiert.
    - Die Chance für die erfolgreiche Anwendung dieser Spezialfähigkeit liegt bei
      - 30 % ab einem Lebenspunkte-Wert von 50 oder weniger
      - 50 % ab einem Lebenspunkte-Wert von 20 oder weniger
      - 70 % ab einem Lebenspunkte-Wert von 10 oder weniger
  - können **Schaden** nehmen (bei einem Angriff)

Weitere Charakter-Typen sollen beliebig hinzugefügt werden können. Diese können immer:

- Angreifen
- Spezialfertigkeit aktivieren
- Spezialfertigkeiten deaktivieren
- Schaden nehmen

Funktionen der Kampfarena

Die Kampfarena hat folgende Funktionen:

- Es können je zwei Rivalen gegeneinander antreten.
- In jeder Runde können die Rivalen entweder angreifen ODER die Spezialfertigkeit aktivieren ODER die Spezialfertigkeit deaktivieren.
- Wer beginnt, wird zufällig festgelegt.
- Die Wahl über die Aktivität pro Runde und Rivale liegt beim Spieler und soll über die Kommandozeile erfolgen.
- Der Kampf endet, wenn einer der Rivalen keine Lebenspunkte mehr hat.
- Während des Kampfes sollen für die Spieler wichtige Statusinformationen ausgegeben werden (z.B. wer ist am Zug, welche Aktion wurde getätigt, welche Auswirkungen hatte die Aktion, wie viele Lebenspunkte haben die Kontrahenten, ist die Spezialfähigkeit aktiviert, etc.).

Ein Kampf soll auf der Kommandozeile über entsprechende Eingaben abgewickelt werden können. Statusmeldungen des Kampfes bzw. die Stats der Charaktere werden auf die Kommandozeile ausgegeben.

Erweiterbarkeit des Systems

Wichtig ist, dass das System leicht mit neuen Charakterklassen (z.B. Dieb, Barde, Barbar etc.) etc. erweitert werden kann, denn weitere Charakterklassen kommen später im Rahmen von kostenpflichtigen DLCs dazu. Alle diese Klassen haben dann immer die gleichen Methoden (siehe oben), die sie allerdings je nach Definition der Charakterklasse ganz unterschiedlich implementieren können. Die Arena soll davon jedoch unbetroffen sein. Die programmierte Arena soll mit jeder möglichen (auch neuen) Charakterklasse ohne Modifikationen funktionieren.

Dokumentierung

- Zeichne vor Beginn ein UML-Diagramm
- Verwende ein GitHub-Repository
- Kommentiere mit JavaDoc-Kommentaren

Abgabe

Gib alle Source-Files über ein GitHub-Repository ab.

Bereits erledigte Aufgaben und Hinweise

Dein Arbeitskollege hat sich schon einige Gedanken zur Umsetzung gemacht:

- Für eine **Zufallszahl** innerhalb eines bestimmten Bereichs min bis max kann in Java folgende Klasse aus java.util.concurrent.ThreadLocalRandom verwendet werden:
  - `ThreadLocalRandom.current().nextInt(min, max + 1);`
- Für die Berechnung der **Eintrittswahrscheinlichkeit für die Spezialfähigkeit der Zwerge** kann man ebenfalls Zufallszahlen verwenden:
  - Zuerst berechnen wir eine Zufallszahl zwischen 1 und 10.
  - Die Spezialfähigkeit wird nun ausgeführt, wenn die Zahl innerhalb eines bestimmten Bereichs ist. Je kleiner der Bereich, desto geringer die Wahrscheinlichkeit.
  - Beispiel:
    - Wir führen die Spezialfähigkeit nur aus, wenn die berechnete Zufallszahl zwischen 1 und 3 ist à 30 % Eintrittswahrscheinlichkeit
    - Wir führen die Spezialfähigkeit nur aus, wenn die berechnete Zufallszahl zwischen 1 und 5 ist à 50 % Eintrittswahrscheinlichkeit
    - Wir führen die Spezialfähigkeit nur aus, wenn die berechnete Zufallszahl zwischen 1 und 7 ist à 70 % Eintrittswahrscheinlichkeit
- Die **Arena** könnte wie folgt aufgebaut sein:
  - Zwei Datenfelder mit den Charakteren, die gegeneinander antreten (werden im Konstruktor gesetzt). Die Kontrahenten werden als Konstruktionsparameter mitgegeben.
  - Ein Datenfeld wird vorgesehen, in dem der Gewinner eingetragen wird, wenn einer feststeht.
  - fight()-Methode:
    - Pro Runde kämpft Charakter 1 gegen Charakter 2 (wer beginnt wird zufällig festgelegt) und dann (falls Charakter 2 den Zug überlebt) Charakter 2 gegen Charakter 1;
    - Der Kampf läuft solange, solange kein Gewinner feststeht.
  - Hilfsmethode simulateCombat(Charakter *attacker*, Charakter *victim*)
    - Den Kampf zwischen Charakter 1 und Charakter 2 kann man über eine Hilfsmethode abwickeln simulateCombat(Charakter *attacker*, Charakter *victim*)
    - In ihr wird von der Kommandozeile abgefragt, welche Aktion durchgeführt werden soll (Angriff, Spezialfähigkeit aktivieren, Spezialfähigkeit deaktivieren)
    - Je nach User-Input werden die verschiedenen Methoden am *attacker* aufgerufen.
    - Wenn das *victim* in einer Runde null oder weniger Lebenspunkte erreicht, wird der *attacker* als Gewinner (Datenfeld) gesetzt.
    - Die fight() Methode beendet den Kampf.
- Eine App-Klasse instanziiert eine Arena mit zwei neuen Charakteren und beginnt über die Methode fight() den Kampf.