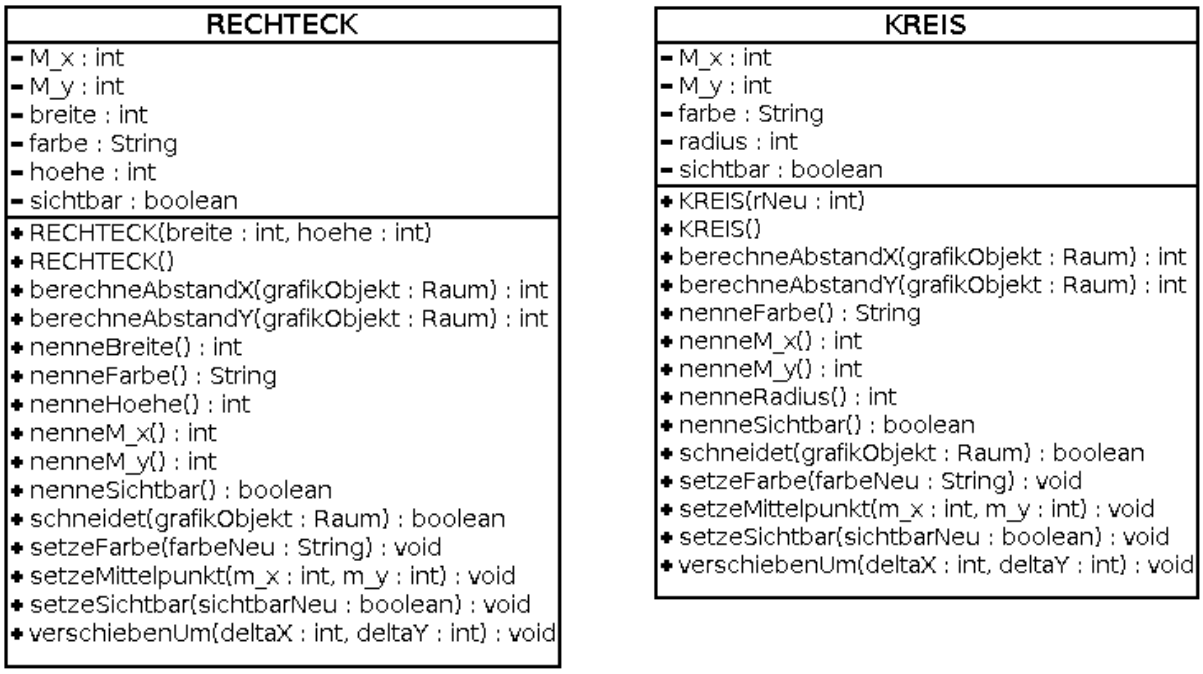
Übungen zur objektorientierten Modellierung

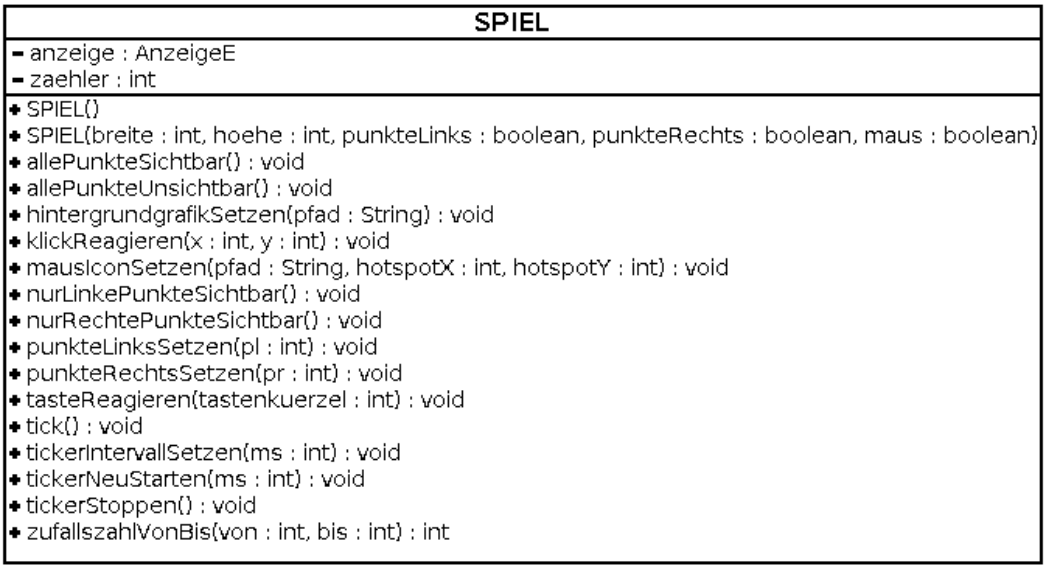
Unter objektorientierter Modellierung versteht man das detailgetreue Darstellen einer zu programmierenden Szene durch Skizzen in UML. UML steht für Unified Modelling Language und beinhaltet unter anderem Klassen-Diagramme, Klassen-Karten, Objekt-Diagramme und Objekt-Karten. Je genauer man sich die reale Situation ansieht und sie anlysiert, um so einfacher lässt sie sich danach in JAVA-Code fassen. Kein Programmierer legt los, ohne vorher ein Modell zu bilden …

Im Folgenden wird dir dieser Modellierungs-Prozess zunächst an einem Beispiel gezeigt. Anschließend sollst du diesen Prozess an weiteren Beispielen selbst aktiv anwenden. Die Grafik wird dir dabei durch Klassen der Engine-Alpha vorgegeben.

Objekt-Karten zu den vorgegebene Klassen

Für die folgende Arbeit werden dir einige Klassen aus der Engine-Alpha vorgegeben. Mithilfe dieser Klassen werden alle grafischen Darstellungen realisiert.

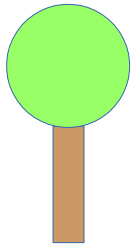




Demonstrations-Beispiel

Ein Baum besteht aus einem Stamm und einer Krone, die natürlich durch Referenz-Attribute grafisch dargestellt werden sollen.

Der gesamte Baum hat eine Höhe (in Metern) und ist von einer bestimmten Art (Ahorn, Buche, …).



Der Baum kann welken: Dabei wird die Krone orange.

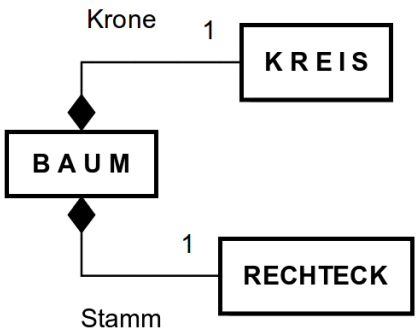
Der Baum kann neu austreiben: Dabei wird die Krone wieder grün.

Zum Messen der Höhe des Baums soll eine entsprechende Methoden vorhanden sein, ebenso um festzustellen, von welcher Art der Baum ist.

Du sollst nun der Reihe nach folgende Fragen durchdenken und deine Antworten in sauberem **UML** (**U**nified **M**odelling **L**anguage) darstellen – **K E I N JAVA-Code !**

1. Klassen-Modell finden

a) Welche **Klassen** werden für diese Aufgabe benötigt?
(eigene und vorgegebene)
Welche **Beziehungen** bestehen zwischen diesen Klassen?
(Referenz-Attribute, Vererbung)
Zeichne ein einfaches Klassen-Diagramm.
Achte jeweils auf die richtigen Symbole,
eine aussagekräftige Benennung sowie Kardinalitäten.



b) Welche **Attribute** und **Methoden** sind für jede dieser Klassen sinnvoll?
Geerbte Attribute und Methoden werden NICHT erneut angegeben.
Brauchen die Methoden Übergabe-Parameter? Geben Sie Antwort?
Fertige für jede eigene Klasse eine ausführliche Klassen-Karte.
Berücksichtige jeweils alle Attribute und deren Daten-Typ sowie alle Methoden, ob sie Übergabe-Parameter brauchen oder Antwort geben.

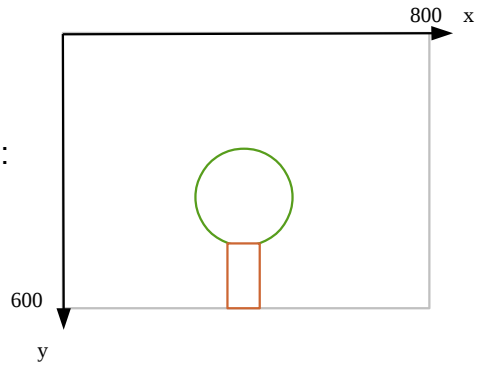
Bei den Methoden interessieren wir uns hier noch nicht dafür, wie sie im Programm umgesetzt werden können, sondern nur, welche es geben soll und deren Struktur.

BAUM
- krone : *KREIS* - stamm : *RECHTECK* - hoehe : *float* - art : *String*
+ BAUM(art : *String*) + nenneArt() : *String* + nenneHoehe() : *float* + welken() : *void* + austreiben() : *void*

2. Objekt-Modell finden

Ein durch seinen Konstruktor neu erzeugter Baum ist von der Art, die im Konstruktor angegeben wird und ist 0,5m groß. Seine Krone ist ein eigenständiges KREIS-Objekt, dessen Lage, Radius und Farbe erst festgelegt werden muss. Der Stamm ist ein eigenständiges RECHTECK-Objekt, dessen Lage, Breite, Höhe und Farbe erst festgelegt werden müssen.

a) Zeichne in einem geeigneten Koordinaten-System einen Baum (Einheit Pixel).



Gib in der Zeichnung vom **Stamm** folgendes in Pixeln an:

Breite = _____

Höhe = _____

Mittelpunkt = _____

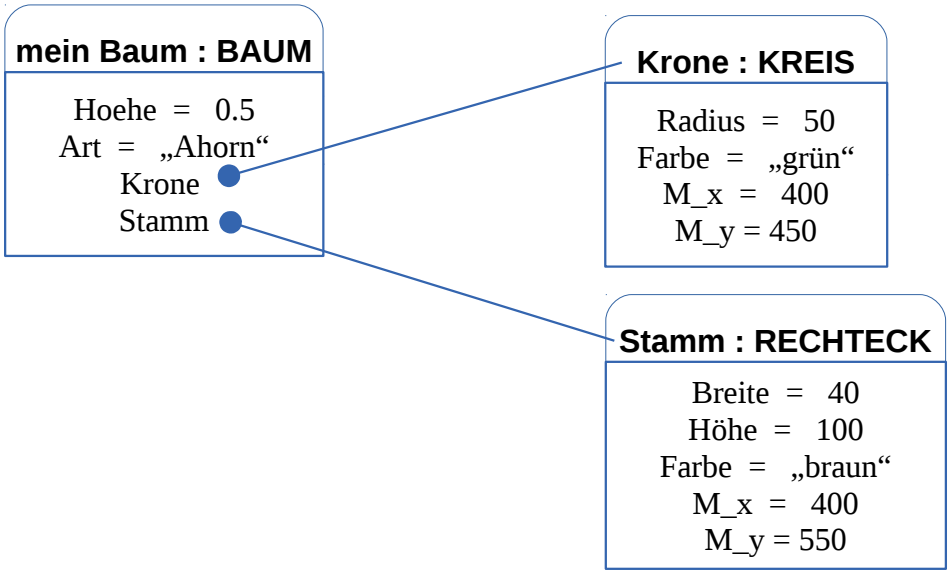
Gib außerdem von der **Krone** folgendes an:

Radius = _____

Mittelpunkt = _____

b) Fertige ein ausführliches Objekt-Diagramm eines frisch erzeugten Baums an.

Denke auch an die Referenz-Attribute *Stramm* und *Krone*. Das sind zwei weitere Objekte.



3. Klassen- und Objekt-Modell in JAVA umsetzen

Du hast nun die neue Klasse BAUM modelliert. Dein Modell enthält auch die Beziehungen dieser Klasse zu den Klassen RECHTECK und KREIS. Du weißt, wie du die Attribute mit Werten zu belegen hast und dass du den Stamm und die Krone als eigenständige Objekte (Referenz-Attribute der Klasse BAUM) gestalten musst.
Lediglich die Rümpfe einiger Methoden sind dir noch unklar. Diese Rümpfe füllen wir vorerst nur mit dem **Kommentar `//ToDo`**.

Nun ist es an der Zeit, das bisherige Modell in ein Programm umzusetzen.
Du sollst also den Baum am Bildschirm grafisch darstellen.

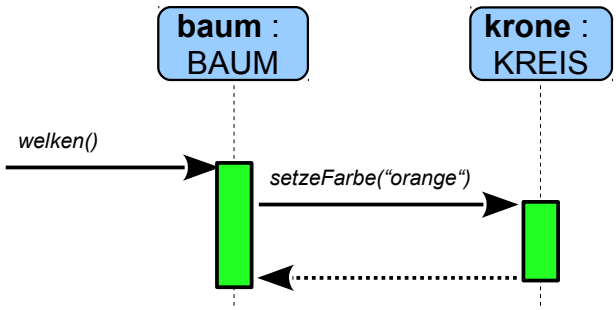
Öffne das *BlueJ-Projekt* ***BAUM_SIMULATION_Vorlage*** .

- → Schreibe eine neue ***Klasse BAUM*** mit genau den ***Attributen***, die du in der Klassen-Karte modelliert hast.
- → Der ***Konstruktor*** soll zuerst die primitiven Attribute initialisieren.
- → Anschließend sollen die Referenz-Attribute als neue Objekte erzeugt werden.
- → Vergiss nicht, weitere Methoden der Stamm- und Krone-Objekte im Konstruktor von BAUM aufzurufen, damit sie im Grafik-Fenster genau so aussehen, wie du es in der vorigen Aufgabe gezeichnet hast.
- → Die ***sondierenden Methoden*** kannst du schon in JAVA umsetzen.
- → Die ***verändernden Methoden*** `welken()` und `austreiben()` legen wir zwar an, schreiben aber im Rumpf zunächst nur den Kommentar `//ToDo`.

4. Sequenz-Diagramme für die Methoden

In sogenannten Sequenz-Diagrammen stellt man dynamisch den ***Ablauf von Methoden*** dar. Insbesondere wird darin der ***zeitliche Verlauf der Kommunikation zwischen den beteiligten Objekten*** dargestellt.

Das ***Sequenz-Diagramm*** rechts veranschaulicht die sehr einfache ***verändernde Methode*** `welken()` der Klasse Baum.

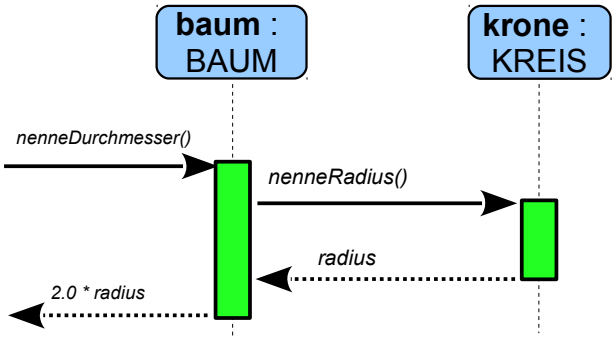


Welkt ein Baum, so stellen wir das durch eine orange statt grüne Krone dar. Wenn also bei unserem Baum-Objekt die Methode `welken()` aufgerufen wird, so muss im Rumpf dieser Methode der Methoden-Aufruf `setzeFarbe("orange")` an das vom Baum referenzierte Krone-Objekt gesendet werden. Der gestrichelte Pfeil zurück signalisiert, dass die Methode des Krone-Objekts fertig ausgeführt ist. Da der gestrichelte Pfeil nicht beschriftet ist, kann man sehen, dass die Methode ***keine Antwort*** gibt.

Das ergibt dann folgenden JAVA-Code in der Klasse BAUM:

```
public void welken() {
    this.krone.setzeFarbe(„orange");
}
```

Sehr ähnlich sähe das ***Sequenz-Diagramm*** einer ***sondierenden Methode nenneDurchmesser()*** aus.



Soll ein Baum-Objekt seinen Kronen-Durchmesser nennen, so fragt es erst sein Krone-Objekt nach dessen Radius (in Pixel).
Die Antwort wird einfach auf den gestrichelten Pfeil geschrieben, der vom Krone-Objekt zum Baum-Objekt zurück geht.
Die so erhaltene Antwort wird vom Baum-Objekt verdoppelt (Durchmesser).
Das Ergebnis dieser Berechnung wird nun an den Aufrufer der Methode `nenneDurchmesser()` weiter gereicht, was man an dem zweiten gestrichelten Pfeil sehen kann.

Hier ergibt sich dann folgender JAVA-Code:

```
public float nenneDurchmesser() {
    return  2.0 * this.krone.nenneRadius();
}
```

Aufgabe

Fertige in ähnlicher Weise das Sequenz-Diagramm zur Methode `austreiben()` an.

Dabei soll die Krone wieder grün dargestellt werden.

Setze die Methode auch in JAVA um.

Schüler-Projekt: Auto

Fertige ein Modell zu einem einfachen Auto an.

Gehe dabei exakt wie in dem obigen Beispiel vor.

Ein Auto hat neben den Referenz-Attributen für die Grafik auch noch die Attribute Kennzeichen und Geschwindigkeit. *(in Pixel pro Bewegungs-Schritt)*

Das Kennzeichen und auch die Geschwindigkeit sollen jeweils neu gesetzt als auch genannt werden können.

Es soll eine Methode `bewegen()` geben, die alle Bauteile des Autos um 2 Pixel nach rechts verschiebt. Hierzu verfügen alle Grafik-Klassen der Engine-Alpha über die Methode `verschiebenUm(… , …)` .

Fertige erst das vollständige Modell an und zeige es dem Lehrer.

Wenn du vom Lehrer das OK bekommst, kannst du dein Modell in JAVA umsetzen – nicht vorher !!!

Zusatz-Aufgabe

Schreibe eine weitere Klasse ANIMATION, die von der Klasse SPIEL erbt.

In dieser Klasse soll es ein Referenz-Attribut vom Typ BAUM geben und eines vom Typ AUTO.

Überschreibe die Methode `tick()` und lasse das Auto fahren.

Überschreibe die Methode `tasteReagieren(...)` und führe bei Tastendruck W die Methode `welken()` aus und bei Taste A die Methode `austreiben()`.