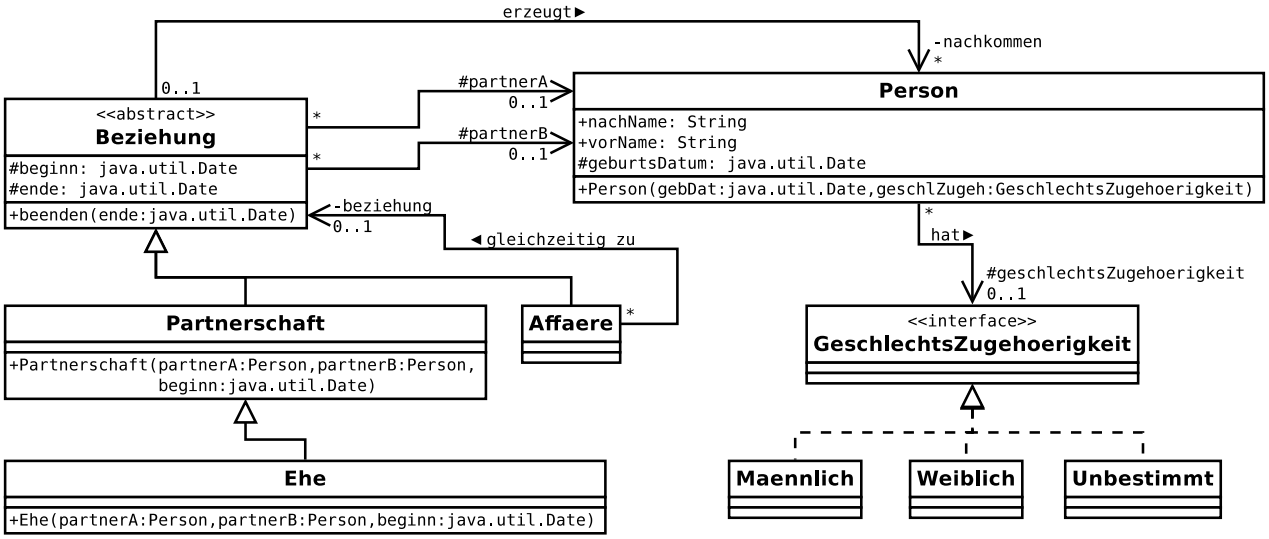
FAU, Informatik 2, AUD-Team
aud@i2.cs.fau.de

6. Übung

Abgabe bis 12.06.2017, 10:00 Uhr

Einzelaufgabe 6.1: Person — 10 EP

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



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. evtl. muss bei Konstruktoren ein gültiger `super`-Aufruf eingefügt werden.

Wichtig: Verschachteln Sie keine Klassen oder Schnittstellen! 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. Verwenden Sie bei *unbeschränkter Multiplizität* hier grundsätzlich Felder!

Einzelaufgabe 6.2: Assoziation **9 EP**

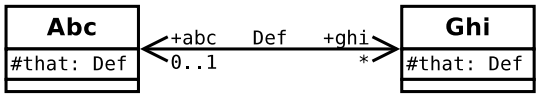
Gegeben sei folgender Java-Quellcode:

```
class Abc {
}

class Def {
    public Abc abc;
    public Ghi[] ghi;
    protected Def that;
}

class Ghi {
}
```

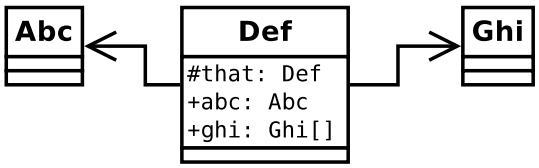
Bestimmen Sie für jedes UML-Diagramm in den folgenden Teilaufgaben, ob das jeweils dargestellte Klassendiagramm zum Quellcode passt *und begründen* Sie Ihre Antworten.

a) „Passt“ oder „Passt nicht“:

b) „Passt“ oder „Passt nicht“:

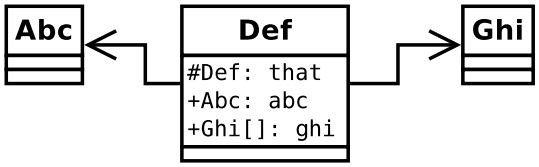


c) „Passt“ oder „Passt nicht“:

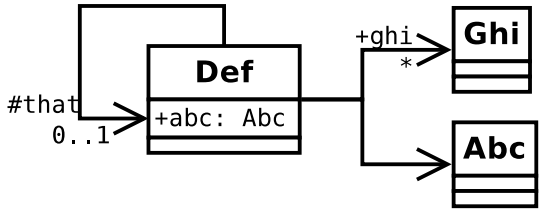


d) „Passt“ oder „Passt nicht“:

e) „Passt“ oder „Passt nicht“:

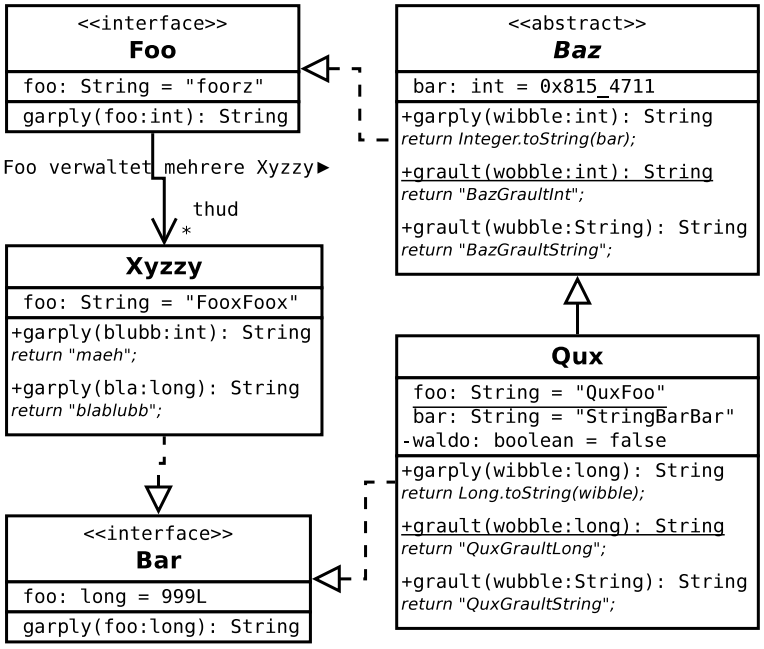


f) „Passt“ oder „Passt nicht“:



Einzelaufgabe 6.3: Polymorphie — 10 EP

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

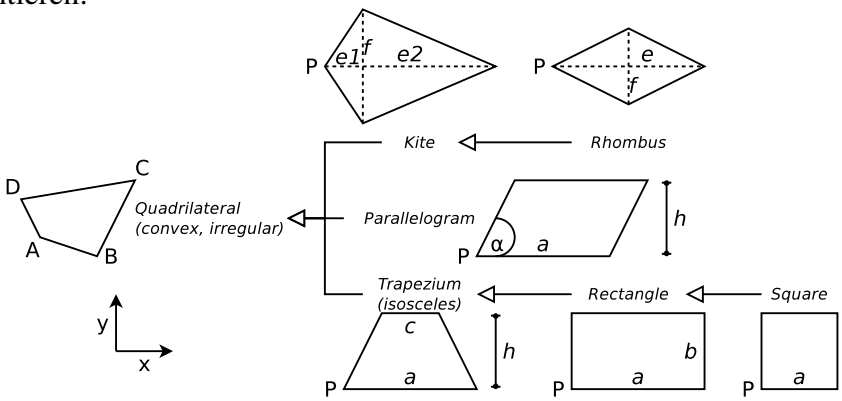


Geben Sie an, welche Ausgaben die mit (1)–(10) gekennzeichneten Zeilen im folgenden Java-Programm erzeugen *und begründen* Sie Ihre Antwort kurz:

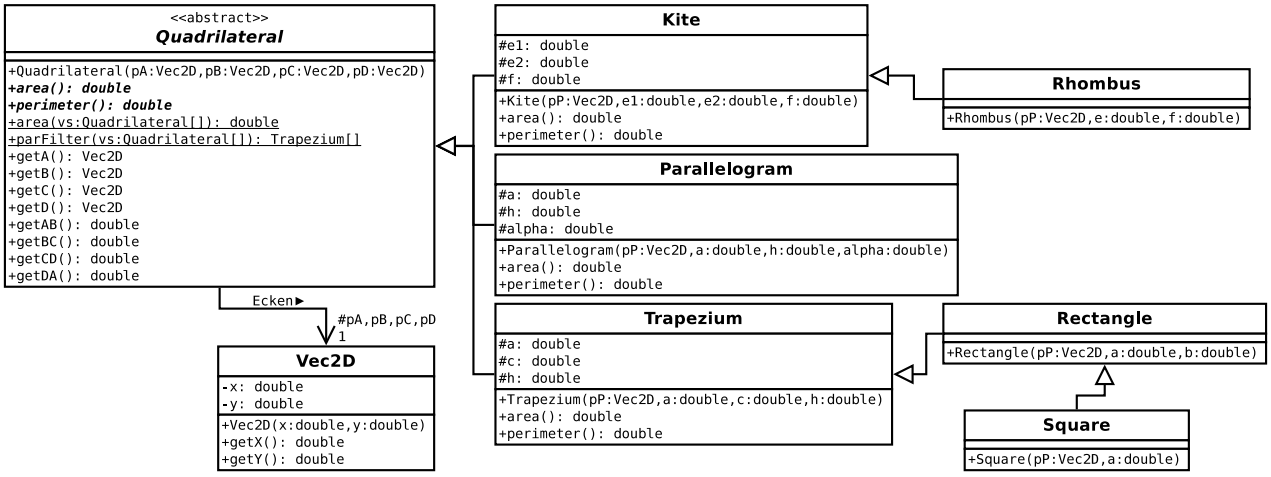
```
public class Polymorphie {
    public static void main(String[] args) {
        Bar rq = new Qux();
        Foo oq = new Qux();
        Baz zq = new Qux();
        Qux xq = new Qux();
        Bar rx = new Xyzzy();

        /* (01) */ print(rq.foo);
        /* (02) */ print(oq.garply(666));
        /* (03) */ print(oq.foo);
        /* (04) */ print(zq.garply(4711));
        /* (05) */ print(zq.foo);
        /* (06) */ print(zq.grault(Integer.MAX_VALUE * 42));
        /* (07) */ print(zq.grault("grault"));
        /* (08) */ print(xq.foo);
        /* (09) */ print(rx.garply(1));
        /* (10) */ print(rq.garply(2));
    }

    public static void print(Object o) {
        System.out.println(o.toString());
    }
}
```

Gruppenaufgabe 6.4: Vererbung und Polymorphie — 31 GP

In dieser Aufgabe sollen Sie Vererbung und Polymorphie an einem praktischen Beispiel üben. Stellen Sie sich dazu vor, Sie müssen für ein Geometrie-Lernprogramm folgende Vierecke als Java-Klassen repräsentieren:



Implementieren Sie zunächst folgendes UML-Klassendiagramm *so genau wie möglich*:



Beachten Sie dazu auch folgende Hinweise:

- Die Java-Klassen müssen *exakt* die im Klassendiagramm deklarierten Attribute und Methoden haben – *nicht mehr und nicht weniger*!
- Die Klasse `Vec2D` stellt Vektoren bzw. Punkte (x, y) in der Ebene dar. Die Konstruktorparameter werden in den gleichnamigen Attributen gespeichert und mittels `get•` bereitgestellt.
- Die Klasse `Quadrilateral` stellt beliebige Vierecke mit den Ecken `pA`, `pB`, `pC` und `pD` (jeweils als `Vec2D`-Objekte) dar. Diese Klasse ist `abstract` und hat zwei `abstract`e Methoden `area` und `perimeter` sowie zwei Klassenmethoden `area` und `parFilter` (siehe unten). Die Ortsvektoren aller Eckpunkte werden hier jeweils über `get•` bereitgestellt. Entsprechend wird die Länge der Kanten $\overline{AB}$, $\overline{BC}$, $\overline{CD}$ bzw. $\overline{DA}$ in `get••` berechnet.
- Jede weitere Viereck-Art ist im Punkt `pP` „verankert“, der in der Oberklasse `Quadrilateral` zum Punkt `A` wird. Alle anderen Ecken werden entgegen dem Uhrzeigersinn benannt.
- Ein `Parallelogram` wird hier durch die Ecke `P`, die Kantenlänge `a` (entspricht der Länge der Kanten $\overline{AB}$ bzw. $\overline{CD}$), die Höhe `h` sowie dem Winkel `alpha` ($\sphericalangle DAB$ in Grad!) festgelegt. Zur Vereinfachung „liegt“ jedes Parallelogramm mit der Kante $\overline{AB}$ parallel zur x-Achse.

- Das `Trapezium` in diesem Programm soll gleichschenklig und symmetrisch sein (d.h. $\overline{BC}$ und $\overline{DA}$ sind gleich lang aber in der Regel nicht parallel). Die beiden parallelen Seiten $\overline{AB}$ (mit Länge a) und $\overline{CD}$ (mit Länge c) liegen zur Vereinfachung ebenfalls parallel zur x-Achse.
- Ein `Rectangle` ist hier ein spezielles Trapez, das nur rechte Winkel hat und dessen Höhe zugleich die Kante `b` ($\hat{=}$ Länge von $\overline{BC}$ bzw. $\overline{DA}$) ist.
- Ein `Square` ist ein spezielles Rechteck, bei dem auch alle Kanten gleich lang (`a`) sind.
- Ein `Kite` hat zwei senkrechte Diagonalen $\overline{AC}$ bzw. $\overline{BD}$ der Länge (`e1` + `e2`) bzw. `f`, wobei $\overline{AC}$ die Diagonale $\overline{BD}$ genau in der Mitte schneidet.
- Ein `Rhombus` ist ein spezielles `Kite`, bei dem beide Diagonalen $\overline{AC}$ bzw. $\overline{BD}$ der Länge `e` bzw. `f` einander jeweils in der Mitte schneiden.
- Die statische Methode `area` der Klasse `Quadrilateral` bekommt ein Feld beliebiger Länge mit unterschiedlichen `Quadrilateral`-Objekten und soll die Summe der Flächeninhalte aller `Quadrilateral`e berechnen.
- Die Klassenmethode `parFilter` der Klasse `Quadrilateral` bekommt ein Feld unterschiedlicher `Quadrilateral`-Objekte und soll daraus *genau* die Objekte in einem Feld passender Länge zurückgeben, die typkompatibel zu `Trapezium` sind – also auch alle Unterklassen wie z.B. `Square`s. Falls es keine passenden Objekte gibt, dann soll die Methode ein leeres Feld zurückgeben (*nicht* `null` und *keine* Ausnahme werfen!).
- Beachten Sie, dass Ihre Implementierungen (insbesondere der Klasse `Quadrilateral` und ihrer beiden Methoden `area` und `parFilter`) mit *beliebigen* `Quadrilateral`en zurecht kommen müssen – auch `Quadrilateral`-Objekte, deren Klassen *nicht* im obigen UML-Diagramm stehen ...

29 EP + 31 GP = 60 Punkte