

Informatik II

Zu jeder Aufgabe sind Dateien abzugeben, deren Namen rechts in der Aufgabenüberschrift stehen. Stellen Sie die Dateien in ein extra Verzeichnis (mit beliebigem Namen) und packen Sie dieses zu einem ZIP-Archiv. Geben Sie dieses, wie üblich, per UniWorx ab.

Aufgabe 6-1

Lebensdauer von Objekten, Gültigkeitsdauer von Variablen
(Animal1.{pdf,ps,jpg,png,gif}, Animal2.*, Animal3.*, 9 Punkte)

```
public class Animal {
    public static int animalCounter = 0;
    private String name;
    public Animal(String myName) {
        this.name = myName;
        animalCounter++;
    }
    public void say(String str) {
        String output = name+" says: "+str;
        System.out.println(output);
        // (2)
    }
    public static void main(String[] args) {
        // (1)
        Animal rex = new Animal("Rex");
        rex.say("woof");
        rex = null;
        // (3)
        Animal kitty = new Animal("Kitty");
        kitty.say("meow");
        System.out.println("Wir haben "+Animal.animalCounter+" Tiere erzeugt.");
    }
}
```

Stellen Sie die Speicherzustände, die zu den untenstehenden Zeitpunkten auftreten, als Bindungs-Stack-Diagramm dar. Bindungs-Stack-Diagramme sind die Visualisierungen der Keller, die wir gemalt haben, wenn es um die Lebensdauer von Variablen ging. Beispiele: Zentralfolien KW 22, Seite 11 oder Folien zur Vererbung, Seite 28. Genau wie in der Zentralübung soll dabei der Stack als verkettete Liste von Frames dargestellt werden.

1. Anfang der `main`-Methode
2. Ende der Methode `say()`, beim Aufruf `rex.say('woof')`
3. Nach der Zuweisung `rex = null`

Hinweise:

- Beachten Sie, dass Stack-Frames automatisch gelöscht werden, Objekte (in dieser Aufgabe) hingegen nicht.
- Der Initialisierungswert von Variablen mit Objekt-Typ ist `null`.

- Arrays und Strings sind auch Objekte, aber hier greift die bisher verwendete Notation (mit Attributen) nicht. Sie können also (innerhalb eines vernünftigen Rahmens) selbst kreativ werden.
- Damit auch den Klassen Rechnung getragen wird, soll in den Diagrammen eine Klassentabelle erscheinen, die wie ein Stack-Frame gezeichnet wird. Sie zeigt auf Klassen, die in der Objekt-Notation (als Instanz der Klasse **Class**) gezeichnet werden.
- Schreiben Sie neben jedes Frame, zu welcher Methode es gehört, oder ob es die Klassentabelle ist.
- Welche Objekte können nach (3) von Garbage-Collector als Müll entsorgt werden?

Aufgabe 6-2

Beschäftigte bei Behörden, Teil II

(*java, Klassendiagramm, {pdf,ps,jpg,png,gif}, 10 Punkte)

Wir ergänzen die Aufgabe 5-1, um neben einfachen Angestellten noch deren Manager modellieren zu können.

Ein Manager ist gegeben durch

- seinen Nach- und Vornamen (jeweils vom Typ **String**)
- das Jahr seiner Geburt (von Typ **int**)
- einem Grundgehalt (vom Typ **int**)
- einen booleschen Wert, der anzeigt, ob der Beschäftigte verheiratet ist.

Das Gehalt eines Managers errechnet sich denkbar einfach: Das Grundgehalt wird um 25% erhöht, wenn der Manager verheiratet ist. Da ein Manager sowieso eine steile Karriere absolviert, ist eine Koppelung an das Alter nicht notwendig.

- a) Zeichnen Sie ein UML-Diagramm, das die Klasse **Clerk**, eine Klasse **Manager** sowie eine Oberklasse **Person** enthält. Überlegen Sie sich dabei, welche Attribute und Methoden in der Oberklasse definiert werden sollen.

Hinweis: Ein praktisches kleines Programm zum Zeichnen von UML-Diagrammen ist UMLet (<http://www.umlet.com/>).

- b) Implementieren Sie die drei Klassen (die Klasse **Clerk** können Sie übernehmen, eine Musterlösung findet sich auf der Webseite).

Hinweis: Wenn man eine Methode mit dem Schlüsselwort **abstract** versieht, wird diese quasi zu einer „Schablone“. Man muss (und kann) dann keinen Rumpf angeben, ist aber beim Erstellen von Unterklassen gezwungen, dort diese Methode zu implementieren. Zudem muss eine Klasse, die eine abstrakte Methode enthält, ebenfalls als **abstract** deklariert werden. Eine abstrakte Klasse kann nicht instanziiert werden, wohl aber ihre nicht-abstrakten Unterklassen. Mit einer abstrakten Methode schafft man also einen Platzhalter in der Oberklasse, ohne eine konkrete Implementierung liefern zu müssen und zwingt gleichzeitig die Unterklassen, diese Methode zu implementieren. Hier ein Beispiel, bei dem die Klasse **Animal** aus Aufgabe 6-1 mit anderen Mitteln implementiert wird:

```
public abstract class AbstractAnimal {
    public void say(String str) {
        String output = getName()+" says: "+str;
        System.out.println(output);
    }
    public abstract String getName();

    public static void main(String[] args) {
        AbstractAnimal rex = new Rex();
        rex.say("woof");
        AbstractAnimal kitty = new Kitty();
        kitty.say("meow");
    }
}
```

```

public class Rex extends AbstractAnimal {
    public String getName() {
        return "Rex";
    }
}

public class Kitty extends AbstractAnimal {
    public String getName() {
        return "Kitty";
    }
}

```

- c) Manager beneiden Manager genauso wie Angestellte Angestellte beneiden. Angestellte jedoch beneiden Manager nur, wenn diese mehr als doppelt so viel verdienen (und jünger sind), während Manager alle Angestellten beneiden, die mehr verdienen, unabhängig vom Alter. Implementieren Sie die geänderte `envies`-Methode für Manager und Angestellte.

Hinweis: Um zu prüfen, ob eine Klasse von einem Typ ist, können Sie mit `<Objekt> instanceof <Klasse>` einen booleschen Ausdruck erzeugen, der dann `true` liefert, wenn die Klasse des Objekts eine Subklasse von `<Klasse>` ist (dabei ist die Subklassenbeziehung reflexiv, d.h. dass z.B. `String` eine Subklasse von `String` ist). Sie brauchen jedoch `instanceof` nicht zwingend, um diese Aufgabe zu lösen.

Beispiele für die Verwendung von `instanceof`:

```

AbstractAnimal k = new Kitty();
boolean b1 = k instanceof Kitty;           // liefert true
boolean b2 = k instanceof AbstractAnimal; // liefert true
boolean b3 = k instanceof Rex;             // liefert false
boolean b4 = k instanceof String;          // Compilerfehler: Ein AbstractAnimal
                                           // kann nie ein String sein

```

- d) Passen Sie die Prozedur `allHappy` so an, dass sie mit einer Reihung von `Person`-Objekten funktioniert. Sie sollten die Prozedur dazu in die Klasse `Person` einfügen.
- e) Erstellen Sie eine `main`-Prozedur in der Klasse `Person`, in der eine Reihung verschiedener Angestellter und Manager erstellen und diese per `allHappy` auf Zufriedenheit testen.

Aufgabe 6-3

Polymorphie

(Polymorphie.txt, 3 Punkte)

In den folgenden zwei Klassen werden zwei Arten von Polymorphie eingesetzt.

```

public class A {
    public void printThis() {
        System.out.println("This: A-Instanz");
    }
    public void printArgument(A a) {
        System.out.println("Argument: A-Instanz");
    }
    public void printArgument(B b) {
        System.out.println("Argument: B-Instanz");
    }
    public static void main(String[] args) {
        A a = new B();
        a.printThis(); // (1)
        a.printArgument(a); // (2)
    }
}

```

```
public class B extends A {  
    public void printThis() {  
        System.out.println("This: B-Instanz");  
    }  
}
```

Erklären Sie für die Stellen (1) und (2) in der Methode `A.main`:

- a) Was wird ausgegeben?
- b) Warum (kurze Erklärung)?
- c) Was für eine Art von Polymorphie liegt vor?

Abgabe: Per UniWorx, bis spätestens Montag, den 12.6.2006 um 9:00 Uhr.