Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. M. Wirsing, M. Hammer, A. Rauschmayer

SS 2006
Übungsblatt 6

Informatik II Musterlösung

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.

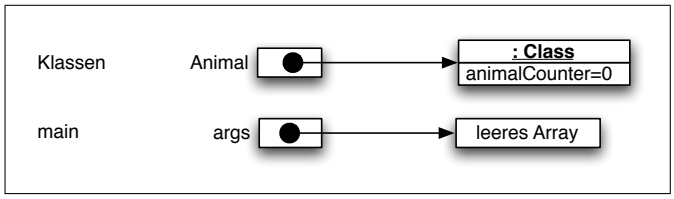
1. Anfang der `main`-Methode
2. Ende der Methode `say()`, beim Aufruf `rex.say(‘‘woof’’)`
3. Nach der Zuweisung `rex = null`

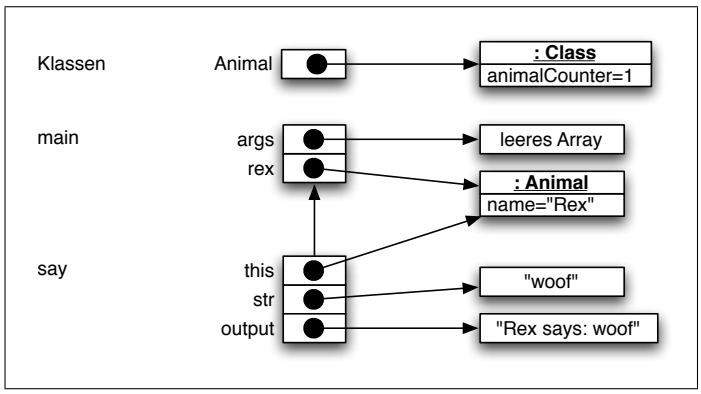
Hinweise:

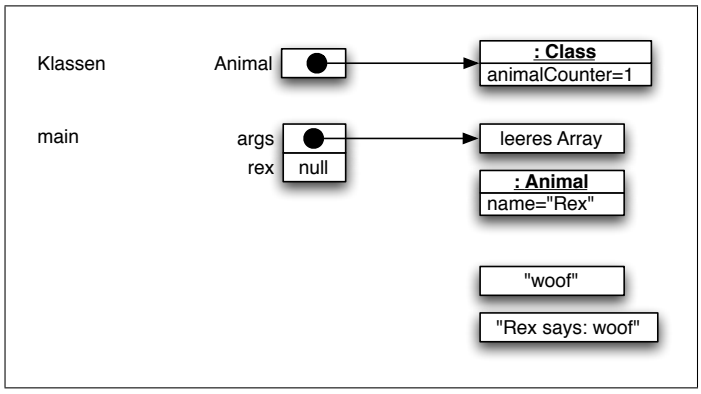
- Beachten Sie, dass Stack-Frames automatisch gelöscht werden, Objekte (in dieser Aufgabe) hingegen nicht.
- Der Initialisierungswert von Variablen mit Objekt-Typ ist `null`.

- 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 dem 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?

Lösung:







Erläuterung: In diesen Stack-Frames schreiben wir nur die lokalen Variablen auf, deren Lebensdauer bereits begonnen hat, die also bereits deklariert werden. Java legt jedoch seine Stack-Frames so an, daß für alle lokalen Variablen bereits Speicher alloziert wird. Es wäre also für konkretes Java korrekt, in den underen beiden Diagrammen jeweils „Kitty“ und „Rex“ im `main`-Stack-Frame zu ergänzen. Im Sinne der Vorlesung sind beide Lösungen sind korrekt: Die oben angegebene Lösung beschreibt eine abstraktere Sicht, bei die Lebensdauer einer lokalen Variable mit ihrer Deklaration beginnt.

Nach (3) können das vorher an `rex` gebundene Objekt `:Animal` und die beiden `String`-Objekte vom Garbage Collector 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

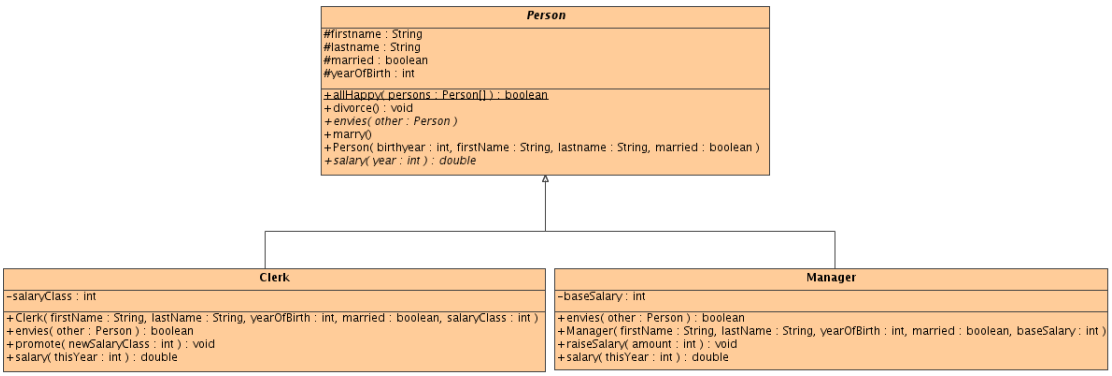
- seinen Nach- und Vornamen (jeweils vom Typ `String`)
- das Jahr seiner Geburt (von Typ `int`)
- einem Grundgehalt (vom Typ `int`)
- einen booleschen Wert, der anzeigt, ob der Beschäftigte verheiratet ist.

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/`).

Lösung:



Erläuterung: Dieses UML-Diagramm verwendet für Attribute die klassische UML-Notation „Name : Type“, während wir in der Vorlesung die mehr an Java angelehnte Variante „Typ Name“ kennengelernt haben. Beide Varianten sind möglich.

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.

Lösung: Klasse `Person`

```
import java.util.Arrays;

/** Diese Klasse enthaelt, was Managern und Angestellten gemein ist: Der Name, das
 * Geburtsjahr und der Verheiratet-Status. Die Methoden, die fuer Manager und
 * Angestellte eine unterschiedliche Funktionalitaet haben, werden als abstrakte
 * Methoden deklariert und damit die Implementierung in den Subklassen gefordert.
 */
public abstract class Person {
    public static final int THIS_YEAR = 2006;
    protected String firstname, lastname;
    protected int yearOfBirth;
    protected boolean married;

    public Person(String firstname, String lastname, int yearOfBirth, boolean married) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.yearOfBirth = yearOfBirth;
```

```
    this.married = married;
}

/** Gibt das Alter in einem gegebenen Jahr zurueck */
public int getAge(int thisYear) {
    return thisYear - yearOfBirth;
}

/** Diese Methode ist von den Subklassen zu implementieren. Sie soll dann,
 * je nach Typ der Subklasse, die Neid-Relation pruefen.
 */
public abstract boolean envies(Person other);

/** toString() (weiter unten) gibt den Namen der Person aus. Das ist fuer
 * Manager und Angestellte gleich. Was sich aendert, ist die Ausgabe des
 * Berufs. Man kann toString() von den beiden Subklassen implementieren
 * lassen, aber hier ist es einfacher, die Subklassen einfach den Beruf
 * angeben zu lassen, was in dieser Methode geschieht.
 */
public abstract String getTypeString();
/** Die Berechnung des Gehalts soll von den Subklassen erledigt werden */
public abstract double salary(int thisYear);

/** Verheiratet werden kann jede Person. Wir pruefen noch, ob sie nicht
 * schon verheiratet ist.
 */
public void marry() {
    assert !married : "Already married!";
    married = true;
}

/** Auch eine Scheidung kann fuer jede verheiratete Person durchgefuehrt werden.
 * Wir pruefen noch, ob sie wirklich verheiratet ist.
 */
public void divorce() {
    assert married : "Cannot divorce a bachelor!";
    married = false;
}

/** toString gibt den Namen aus, und den Beruf, der per getTypeString() von
 * der Subklasse geholt wird.
 */
public String toString() {
    return firstname + " " + lastname + ", " + getTypeString();
}

/** Die allHappy()-Implementierung von Blatt 5 */
public static boolean allHappy(Person[] pers) {
    for (Person p1 : pers) {
        for (Person p2 : pers) {
            if (p1.envies(p2)) {    // beachte die Irreflexitivtaet von envies
                return false;
            }
        }
    }
    return true;
```

```
    }

    /** Eine main-Methode, die mit Math.random() (gibt einen Zufallswert zwischen
     * 0 und 1 als double zurueck) ein Array von Managern und Angestellten initialisiert
     * und schaut, ob zufaellig alle gluecklich sind
     */
    public static void main(String[] args) {
        final int PERSONS = 10;
        Person[] p = new Person[PERSONS];
        for (int i=0; i < PERSONS; i++) {
            if (i % 2 == 0) {
                p[i] = new Clerk("Clerk", "Nr. " + (i/2+1),
                        (int)(1960 + Math.random() * 25.0),
                        Math.random() > 0.3, (int)(Math.random() * 10 + 1));
            } else {
                p[i] = new Manager("Manager", "Nr. " + i/2,
                        (int)(1950 + Math.random() * 25.0),
                        Math.random() > 0.7, (int)(Math.random() * 10000 + 3000));
            }
        }
        System.out.println(Arrays.toString(p) + " is " + (allHappy(p)?"happy":"unhappy"));
    }
}
```

Klasse `Clerk`

```
/** Die Klasse vom Uebungblatt 5. Beachten Sie die Korrektur der Salary-Berechnung. */
public class Clerk extends Person {
    private int salaryClass;
    private static final double basicSalary = 2000.0;
    private static final double classMultiplier = 17.0 / 9.0;
    private static final double marriageBonus = 12.3;

    /** Von den Parametern werden alle bis auf salaryClass an die Superklasse
     * weitergegeben. SalaryClass wird in dieser Klasse gespeichert, da es nur
     * Clerks betrifft.
     */
    public Clerk(String firstname, String lastname, int yearOfBirth,
            boolean married, int salaryClass) {
        super(firstname, lastname, yearOfBirth, married);
        this.salaryClass = salaryClass;
    }


    public boolean envies(Person other) {
        if (other instanceof Clerk) {
            return this.salary(Person.THIS_YEAR) < other.salary(Person.THIS_YEAR)
                && this.getAge(Person.THIS_YEAR) > other.getAge(Person.THIS_YEAR);
        }
        if (other instanceof Manager) {
            return this.salary(Person.THIS_YEAR) < (other.salary(Person.THIS_YEAR) / 2)
            && this.getAge(Person.THIS_YEAR) > other.getAge(Person.THIS_YEAR);
        }
        assert false : "Unbekannter Personentyp " + other.getClass().getName();
        return false;
    }
```

```
    /** Die toString-Methode in Person gibt bereits den Namen aus. Alles, was
     * sich in dieser Klasse an der Ausgabe aendert, ist die Job-Bezeichnung.
     * Anstatt also hier toString() zu ueberladen (was natuerlich auch geht),
     * geben wir einfach nur einen String zurueck, der den Job angibt. Die
     * toString()-Methode wird in Person implementiert.
     */
    public String getTypeString() {
        return "Angestellter";
    }

    public void promote(int newSalaryClass) {
        salaryClass = newSalaryClass;
    }

    /** Berechnet das Gehalt fuer ein gegebenes Jahr */
    public double salary(int thisYear) {
        /* berechnet den Sold */
        double thesalary;
        /* Grundsold + x * 17/9 Prozent mit x = salaryClass: */
        thesalary = basicSalary * ((100.0 + classMultiplier * salaryClass) / 100.0);
        // System.out.println("s1: " + thesalary);
        /* Altersbonus */
        int age = thisYear - yearOfBirth;
        int nbonus = 0;
        if (age >= 30)
            nbonus = (age - 25) / 5;
        thesalary = ((100.0 + nbonus) / 100.0) * thesalary;
        /* Verheiratetenzuschlag */
        if (married)
            thesalary += (marriageBonus / 100.0) * basicSalary;
        return thesalary;
    }
}
```

Klasse Manager

```
/** Ein Manager erbt von Person, wo die Daten wie Name, Alter etc. gespeichert
 * werden. Ein Manager unterscheidet sich nur dadurch, dass er ein Grundgehalt hat,
 * von einem Clerk; ausserdem wird sein Gehalt anders berechnet und er neidet
 * anders. Diese Funktionalitaet wird in der Klasse implementiert.
 *
 * Fuer Kommentare der einzelnen Methoden siehe die Klasse Clerk
 *
 */
public class Manager extends Person {
    private int baseSalary;

    public Manager(String firstname, String lastname, int yearOfBirth,
            boolean married, int baseSalary) {
        super(firstname, lastname, yearOfBirth, married);
        this.baseSalary = baseSalary;
    }

    public boolean envies(Person other) {
        if (other instanceof Clerk) {
```

```
            return this.salary(Person.THIS_YEAR) < other.salary(Person.THIS_YEAR);
        }
        if (other instanceof Manager) {
            return this.salary(Person.THIS_YEAR) < other.salary(Person.THIS_YEAR)
            && this.getAge(Person.THIS_YEAR) > other.getAge(Person.THIS_YEAR);
        }
        assert false : "Unbekannter Personentyp " + other.getClass().getName();
        return false;
    }


    public String getTypeString() {
        return "Manager";
    }

    public void raiseSalary(int amount) {
        baseSalary += amount;
    }

    public double salary(int thisYear) {
        return baseSalary + (married?(double)baseSalary/4.0:0.0);
    }
}
```

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?

Lösung:

1.
   - Ausgabe: `This: B-Instanz`
   - Grund: Wegen der dynamischen Bindung wird zur Laufzeit erkannt, dass in Variable `a` eine Instanz von `B` steckt.
   - Vererbungs-Polymorphie
2.
   - Ausgabe: `Argument: A-Instanz`
   - Grund: Welche Methode zu verwenden ist, wird zur Übersetzungszeit bestimmt und da ist die Variable `a` vom Typ `A`.
   - Überladen

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