University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatik I – EProg HS13

Zwischentest II

Allgemeine Hinweise:

- Die Aufgabe muss vollständig bearbeitet werden, damit Sie die volle Punktzahl erhalten. Insgesamt können **60 Punkte** erreicht werden.
- Die Bearbeitungszeit beträgt **60 Minuten**.
- Unterlagen sind grundsätzlich **keine** erlaubt. Als Ausnahme gilt, falls Sie nicht deutscher Muttersprache sind, ein entsprechendes Wörterbuch.
- Schreiben Sie Ihren **Namen** und Ihre **Matrikelnummer** bitte in die dafür vorgesehenen Kästchen am Ende **jeder** Seite.
- Die Verwendung unerlaubter Hilfsmittel oder das Abschreiben von eine(r/m) Komiliton(in / en) hat die sofortige Abgabe und das Nichtbestehen der Veranstaltung zur Folge. Zudem ist mit einem Disziplinarverfahren zu rechnen.
- Eine englische Version dieses Zwischentestes ist auf Nachfrage verfügbar. Bei Differenzen zwischen der deutschen und der englischen Version dieses Zwischentestes ist die deutsche Version massgebend.

Ich bestätige mit meiner Unterschrift, dass ich:

- die obenstehenden Hinweise gelesen und verstanden habe.
- die Prüfung unter zumutbaren Bedingungen geschrieben habe.

Unterschrift: _____ Datum: _____

Punkte
/60

1 Aufgabe: Multiple Choice (8 Punkte)

Kreuzen Sie Zutreffendes an. Zu jeder Frage können mehrere Antworten richtig sein. Bitte beachten Sie, dass Ihnen für jedes falsch gesetzte Kreuz innerhalb einer Teilaufgabe gleich viele Punkte abgezogen werden, wie Sie für ein korrektes Ankreuzen erhalten. Negative Punktzahlen ergeben null Punkte für die betreffende Frage.

1. Richtige Aussagen sind anzukreuzen: **2 Punkte**
   - ☐ Eine abstrakte Java Klasse kann einen oder mehrere Konstruktor(en) definieren.
   - ☐ Eine abstrakte Java Klasse kann keinen Konstruktor definieren.
   - ☐ Der Konstruktor einer abstrakten Klasse darf keine Parameter definieren.
   - ☐ Der Konstruktor einer abstrakten Klasse kann beliebig viele Parameter definieren.

   Lösung:

   - ☒ Eine abstrakte Java Klasse kann einen oder mehrere Konstruktor(en) definieren.
   - ☐ Eine abstrakte Java Klasse kann keinen Konstruktor definieren.
   - ☐ Der Konstruktor einer abstrakten Klasse darf keine Parameter definieren.
   - ☒ Der Konstruktor einer abstrakten Klasse kann beliebig viele Parameter definieren.

2. Richtige Aussagen sind anzukreuzen: **2 Punkte**
   - ☐ String ist ein Referenz-Datentyp.
   - ☐ Immutable bedeutet, dass eine Variable vom Typ String nie `null` sein kann.
   - ☐ Mit dem Multiplikations-Operator (`*`) können Strings konkateniert werden.
   - ☐ Mit dem Plus-Operator (`+`) können Strings konkateniert werden.

   Lösung:

   - ☒ String ist ein Referenz-Datentyp.
   - ☐ Immutable bedeutet, dass eine Variable vom Typ String nie `null` sein kann.
   - ☐ Mit dem Multiplikations-Operator (`*`) können Strings konkateniert werden.
   - ☒ Mit dem Plus-Operator (`+`) können Strings konkateniert werden.

3. Richtige Aussagen sind anzukreuzen: **2 Punkte**
   - ☐ Eine ”Has-a” Beziehung wird mittels Vererbung modelliert.
   - ☐ Eine ”Has-a” Beziehung wird mittels Instanzvariablen modelliert.
   - ☐ Eine ”Has-a” Beziehung setzt mindestens zwei verschiedene Klassen voraus.
   - ☐ Eine Klasse kann auch zu sich selbst eine ”Has-a” Beziehung haben.

   Lösung:

   - ☐ Eine ”Has-a” Beziehung wird mittels Vererbung modelliert.
   - ☒ Eine ”Has-a” Beziehung wird mittels Instanzvariablen modelliert.
   - ☐ Eine ”Has-a” Beziehung setzt mindestens zwei verschiedene Klassen voraus.
   - ☒ Eine Klasse kann auch zu sich selbst eine ”Has-a” Beziehung haben.

4. Richtige Aussagen sind anzukreuzen: **2 Punkte**
   - ☐ Zwei Methoden die in einer Java Klasse definiert werden, dürfen nicht den gleichen Namen haben.
   - ☐ Zwei Methoden, die in einer Java Klasse definiert werden, den selben Namen haben aber unterschiedliche Parameter erwarten, überladen einander.
   - ☐ Zwei Methoden, die in einer Java Klasse definiert werden, den selben Namen haben aber unterschiedliche Parameter erwarten, überschreiben einander.
   - ☐ Zwei Methoden die in einer Java Klasse definiert werden dürfen die gleichen Parameter erwarten.

Lösung:

- ☐ Zwei Methoden die in einer Java Klasse definiert werden, dürfen nicht den gleichen Namen haben.
- ☒ Zwei Methoden, die in einer Java Klasse definiert werden, den selben Namen haben aber unterschiedliche Parameter erwarten, überladen einander.
- ☐ Zwei Methoden, die in einer Java Klasse definiert werden, den selben Namen haben aber unterschiedliche Parameter erwarten, überschreiben einander.
- ☒ Zwei Methoden die in einer Klasse definiert werden dürfen die gleichen Parameter erwarten.

2 Aufgabe: Rekursion (12 Punkte)

a) **6 Punkte**

Die Fakultät ist eine mathematische Funktion, die einer Zahl das Produkt aller Zahlen kleiner und gleich dieser Zahl zuordnet. Die Fakulät von 5 ist beispielsweise: $5! = 5 * 4 * 3 * 2 * 1 = 120$.

Schreiben Sie eine **rekursive** Methode die die Fakultät einer beliebigen positiven natürlichen Zahl berechnet. Hinweis: Die Fakultät von 0 ist 1.

Ihre Lösung:

```
public int faculty(int number) {
```

Lösung:

```
1 public int factorial(int number) {
2     if (number == 0) {
3         return 1;
4     } else {
5         return number * factorial(number - 1);
6     }
7 }
```

Listing 1: `Rekursion.java`

b) **6 Punkte**

Die Quersumme einer Zahl ist die Summe ihrer Ziffern. Die Quersumme von 42 ist beispielsweise $4 + 2 = 6$. Schreiben Sie eine **rekursive** Methode, die die Quersumme einer beliebigen positiven natürlichen Zahl berechnet.

Ihre Lösung:

```
public int checksum(int number) {
```

Lösung:

```
1 public int checksum(int number) {
2     if (number <= 9) {
3         return number;
4     } else {
5         return number % 10 + checksum(number / 10);
6     }
7 }
```

Listing 2: `Rekursion.java`

3 Aufgabe: Implementation (40 Punkte)

Bilden Sie den unterstehenden Sachverhalt auf (wo angebracht abstrakte) Klassen, Interfaces, Attribute und Methoden ab. Beachten Sie hierzu unbedingt die Implementierungshinweise/-vorschriften weiter unten.

Die Leitung eines Tierheims hat Sie gebeten, bei der Modellierung der neuen Software, die zur Verwaltung der aufgenommenen Tiere benutzt wird, zu helfen. Im Tierheim leben viele verschiedene Tiere (*Animal*). Im Moment sind dies Katzen (*Cat*), Hunde (`Dog`) und Papageien (*Parrot*). Es ist aber denkbar, dass in Zukunft noch weitere Tiere dazukommen werden. Jedes dieser Tiere besitzt eine eindeutige Identifikationsnummer (*ID*). Zusätzlich verbraucht jedes Tier in seinem Leben durchschnittliche eine bestimmte Menge an Nahrungsmittel, die in Kilokalorien angegeben wird (*appetite*). Bei Hunden liegt dieser Nahrungsverbrauch bei 10'000'000 kcal, bei Katzen bei 3'000'000 kcal und die genügsamen Papageien brauchen nur 1'000'000 Kilokalorien. Im Tierheim kann es auch ziemlich laut werden: Hunde bellen (*bark*), Katzen miauen (*miaow*) und Papageien wiederholen alle Sätze die man zu ihnen sagt (*talk*).

Um den Nahrungsbedarf der Tiere zu stillen, kann man alle Tiere mit einer Futterration, die auch in Kilokalorien angegeben wird, füttern (*feed*). Dies führt dazu, dass der Nahrungsbedarf des gefütterten Tieres entsprechend sinkt. Allerdings fressen nur Hunde die ganze Portion. Papageien sind nicht sehr ausdauernd und beenden ihre Mahlzeit immer sobald sie $2/3$ der Portion gefressen haben. Die wählerischen Katzen dagegen rühren das Futter nur bei jeder zweiten Fütterung überhaupt an. In diesem Fall fressen sie aber die gesamte Ration.

Mit einigen Tieren im Tierheim kann man spielen. Wenn mit Hunden gespielt wird, dann beginnen diese mit dem Schwanz zu wedeln (*waggle*). Katzen drücken ihre Zufriedenheit, dass mit ihnen gespielt wird, durch Schnurren aus (*purr*). Papageien sind nicht am Spielen interessiert.

Implementierungshinweise/-vorschriften

- Gehen Sie bei Ihrer Implementierung von folgendem Interface aus:

```
public interface Playful {
   public void play();
}
```

- Überschreiben Sie die *toString* Methode in der Klasse Animal, so dass diese die ID des Tieres zurückgibt.
- Die Werte von Instanzvariablen sollen nur übr Methoden verändert und gelesen werden können.
- Vermeiden Sie Codeduplizität.
- Implementieren Sie einen TestDriver und füttern Sie einen Hund, eine Katze und einen Papagei 10 Mal mit jeweils 200 Kilokalorien Futter. Lassen Sie die Katze und den Hund spielen.

Lösung:

```java
public abstract class Animal {

    private double appetite;

    private int id;

    private static int idCounter = 0;

    public Animal(double appetite) {
        this.appetite = appetite;
        id = idCounter++;
    }

    public abstract void feed(double amount);

    public double getFoodRequirement() {
        return appetite;
    }

    public void setFoodRequirement(double appetite) {
        this.appetite = appetite;
    }

    @Override
    public String toString() {
        return "" + id;
    }

}
```

Listing 3: `Animal.java`

```java
public class AnimalShelter {

    public static void main(String[] args) {
        Dog dog = new Dog();
        Cat cat = new Cat();
        Parrot parrot = new Parrot();

        for (int i = 0; i < 10; i++) {
            dog.feed(10);
            cat.feed(10);
            parrot.feed(10);
        }

        dog.play();
        cat.play();
    }

}
```

Listing 4: `AnimalShelter.java`

```java
public class Cat extends Animal implements Playful {

    private boolean eat = true;

    public Cat() {
        super(3000000);
    }

    @Override
    public void feed(double amount) {
        if (eat) {
            setFoodRequirement(getFoodRequirement() - amount);
        }
        eat = !eat;
    }

    @Override
    public void play() {
        System.out.println("Purr, Purr!");
    }

    public void miaow() {
        System.out.println("Miaow!");
    }

}
```

Listing 5: `Cat.java`

```java
public class Dog extends Animal implements Playful {

    public Dog() {
        super(10000000);
    }

    @Override
    public void feed(double amount) {
        setFoodRequirement(getFoodRequirement() - amount);
    }

    @Override
    public void play() {
        System.out.println("Waggle!");
    }

    public void bark() {
        System.out.println("Bark, bark");
    }

}
```

Listing 6: `Dog.java`

```java
public class Parrot extends Animal {

    public Parrot() {
        super(1000000);
    }

    @Override
    public void feed(double amount) {
        setFoodRequirement(getFoodRequirement() - ((2 / 3) * amount));
    }

    public void talk(String text) {
        System.out.println(text);
    }

}
```

Listing 7: `Parrot.java`

```java
public interface Playful {

    public void play();

}
```

Listing 8: `Playful.java`