## Aufgabe 1 (Programmieren mit Smalltalk)

**5 + 5 Punkte**

Gegeben ist der folgende Nachrichtenausdruck, bei dem eine *unäre* Nachricht an ein Objekt verschickt wird:

```
1    #abc printString
```

a) Geben Sie zwei weitere (beliebige) Nachrichtenausdrücke, wie im Folgenden beschrieben, in Smalltalk an:
   i. einen Nachrichtenausdruck, in dem eine *binäre* Nachricht an ein Objekt verschickt wird.
   ii. einen Nachrichtenausdruck, in dem eine *Schlüsselwortnachricht* an ein Objekt verschickt wird.

**Lösung:**

Nachrichtenausdruck binär:

```
9    1+2
```

Nachrichtenausdruck Schlüsselwortnachricht:

```
10   einObjekt tueEtwasMit: einemParameter und: zweitemParameter
```

b) Benennen Sie bei den drei Nachrichtenausdrücken (die sich insgesamt aus der Aufgabenstellung und aus Aufgabenteil a ergeben), falls vorhanden, jeweils

- Nachricht/Nachrichtenselektor
- Empfängerobjekt
- Parameter

**Lösung:**

`#abc` = Empfängerobjekt
`printString` = Nachricht/Nachrichtenselektor

1 = Empfängerobjekt
+ = Nachricht/Nachrichtenselektor
2 = Parameter

`einObjekt` = Empfängerobjekt
`tueEtwasMit:und:` = Nachricht/Nachrichtenselektor
`einemParameter`, `zweitemParameter` = Parameter

## Aufgabe 2 (Programmieren mit Smalltalk)

**5 + 2 Punkte**

a) Geben Sie in Smalltalk eine Methode mit dem Namen `max` an, die folgende Bedingungen erfüllt: `max` soll einen Parameter übergeben bekommen, den Wert des übergebenen Parameters mit dem Empfängerobjekt vergleichen, und den jeweils größeren Wert zurück geben.

**Lösung:**

```
1    max: aMagnitude
2       |v|
3       v := aMagnitude.
4       self > v
5          ifTrue: [^self]
6          ifFalse: [^v]
```

b) Benennen Sie alle formalen Parameter Ihrer Methode.

**Lösung:**

In der Methodensignatur der Methode `max` ist der formale Parameter `aMagnitude` zu finden.

## Aufgabe 3 (Generalisierung/Spezialisierung/Vererbung)

**5 + 5 Punkte**

a) Wird durch die Verwendung von Vererbung bei der Definition von Klassen immer auch eine Generalisierungs- bzw. Spezialisierungsbeziehung zwischen diesen Klassen geschaffen? Begründen Sie Ihre Aussage.

**Lösung:**

Nein, denn man kann auch Vererbungshierarchien von Klassen konstruieren, die mit dem theoretischen Konzept der Generalisierung/Spezialisierung nichts gemein haben. Unter Vererbung versteht man in der objektorientierten Programmierung die Übertragung der Definition von Eigenschaften und Verhalten (Intension) von einer Klasse auf eine andere. Vererbung dient vor allem der Wiederverwendung von Code und damit der Ökonomie in der Softwareentwicklung. Ob die Vererbung so eingesetzt wird, dass damit auch eine Generalisierungs-/Spezialisierungsbeziehung geschaffen wird, liegt im Ermessen der Programmiererin.

b) Gegeben seien die unten (s. auch nächste Seite) aufgeführten Klassen `Rechteck` und `Parallelogramm`. Ist es möglich, Vererbung einzusetzen? Wenn ja, erläutern Sie, wie die Vererbung eingesetzt werden kann, und welche Vorteile sich ergeben. Wenn nein, erläutern Sie, warum das nicht geht.

| Klasse | Rechteck |
|---|---|
| benannte Instanzvariablen | laenge breite |
| Instanzmethoden | |

```
2     flaeche
3         ^ laenge * breite
4
5     umfang
6         ^ laenge + breite * 2
```

| Klasse | Parallelogramm |
|---|---|
| benannte Instanzvariablen | laenge breite hoehe |
| Instanzmethoden | |

```
7       flaeche
8           ^ laenge * hoehe
9
10      umfang
11          ^ laenge + breite * 2
```

**Lösung:**

Man kann Vererbung einsetzen. Sinnvollerweise erbt `Parallelogramm` von `Rechteck`. Der Vorteil der sich daraus ergibt, ist, dass `Parallelogramm` die Instanzvariablen `laenge` und `breite` und die Instanzmethode `umfang` nicht neu zu definieren braucht. Die Instanzmethode `flaeche` muss jedoch überschrieben werden.

## Aufgabe 4 (Typsysteme)

**6 Punkte**

Warum typisiert man Variablen und andere Programmelemente? Nennen Sie drei Gründe/Vorteile.

**Lösung:**

1. Typisierung regelt das Speicher-Layout.
2. Typisierung erlaubt die effizientere Ausführung eines Programms.
3. Typisierung erhöht die Lesbarkeit eines Programms.
4. Typisierung ermöglicht das automatische Finden von logischen Fehlern in einem Programm (statische/dynamische Typprüfung).
5. Modularisierung von Programmen wenn ein Typ zugleich eine Schnittstelle oder ein Interface ausdrückt.

## Aufgabe 5 (Typsysteme)

**4 + 3 Punkte**

a) Erläutern Sie den Unterschied zwischen statischer und dynamischer Typprüfung.

**Lösung:**

Mit Hilfe der statischen Typprüfung soll die Typkorrektheit zur Übersetzungszeit (vom Compiler) gewährleistet werden. Die dynamische Typprüfung versucht Typkorrektheit zu gewährleisten, indem zur Laufzeit vor einer Variablenzuweisung geprüft wird, ob der zuzuweisende Wert den von der Variable geforderten Typ hat.

b) Nennen Sie den Nachteil der statischen und den Nachteil der dynamischen Typprüfung.

**Lösung:**

Nachteil der dynamischen Typprüfung ist, dass diesbezügliche Fehler erst zu einem Zeitpunkt, in dem man bereits nicht mehr viel anderes machen kann als einen Fehler zu signalisieren, entdeckt werden. Nachteil der statischen Typprüfung ist hingegen, dass durch eine rein statische (und kompromisslos eingesetzte) Typprüfung immer auch Programme zurückgewiesen werden, die nützlich, sinnvoll und typkorrekt sind.

## Aufgabe 6 (Typkonformität)

**4 Punkte**

Gegeben seien die folgenden Klassendefinitionen:

| Typ | A |
|---|---|

| Typ | B |
|---|---|
| Supertyp | A |

Sind bei Vorliegen folgender Variablendeklarationen

```
12    | a A b B |
```

die folgenden Zuweisungen bei geforderter nominaler Typkonformität zulässig?

```
13    a := b
```

```
14    b := a
```

Wie verhält es sich bei geforderter struktureller Typkonformität? Welche der beiden genannten Zuweisungen sind dann zulässig?

**Lösung:**

Die Zuweisung

```
7     a := b
```

Ist sowohl bei geforderter nominaler, als auch bei geforderter struktureller Typkonformität zulässig. Die zweite Zuweisung ist nur bei geforderter struktureller Typkonformität zulässig.

```
8     b := a
```

## Aufgabe 7 (Variablen)

**4 Punkte**

In Sprachen wie Smalltalk und Java ist es schwierig, wenn nicht gar unmöglich, eine Methode zu definieren, die den Inhalt zweier Variablen vertauscht. Erläutern Sie, warum.

**Lösung:**

Sowohl in Smalltalk als auch in Java haben Variablen zwar Referenzsemantik (in Java nur bei Objekttypen), aber beim Methodenaufruf wird nach dem Prinzip des Call by value eine Kopie des Objektpointers übergeben. Zum Vertauschen wäre aber notwendig, dass Zeiger auf die Variablen übergeben werden, so dass der tatsächliche Variableninhalt geändert werden kann. Das ist aber in beiden Sprachen nicht vorgesehen. In C# und C++ ist es hingegen möglich.

## Aufgabe 8 (Abstrakte Klassen)

**4 Punkte**

Was versteht man unter einer abstrakten Klasse und wofür wird sie eingesetzt?

**Lösung:**

Eine abstrakte Klasse ist eine Klasse, von der man keine Instanzen bilden kann. Sie wird eingesetzt, um Generalisierungen innerhalb von Programmcode auszudrücken.

## Aufgabe 9 (Überladen von Methodennamen)

**4 Punkte**

Erläutern Sie, was man unter der Überladung von Methodennamen versteht, und grenzen Sie den Begriff gegen den des Überschreibens ab.

**Lösung:**

In Sprachen wie Java und C++ kann eine Klasse mehrere Methoden gleichen Namens, aber mit verschiedenen Parametertypen besitzen. Solche Methoden nennt man dann überladen. Auch Subklassen können neue, überladene Methoden definieren. Während beim Überladen eine *neue* Methode eingeführt wird, wird beim Überschreiben eine bereits bestehende redefiniert.

## Aufgabe 10 (Interfaces)

**6 + 6 Punkte**

a) Erläutern Sie den Zusammenhang zwischen Interfaces und dem Implementationsgeheimnis (Geheimnisprinzip).

**Lösung:**

Interfaces sind die öffentlichen Schnittstellen von Klassen/Objekten und erlauben den Zugriff auf das jeweilige Objekt. Alles was nicht zum Interface einer Klasse/eines Objekts gehört, wird nach außen verborgen und stellt das Implementationsgeheimnis dar. Damit wird das Geheimnisprinzip, welches ein gängiges Prinzip der objektorientierten Programmierung darstellt, und besagt, dass Implementierungen hinter Schnittstellen (oder Interfaces) zu verbergen sind und nur die Elemente einer Klassendefinition, die für Benutzerinnen einer Klasse zu Verwendung gedacht sind, durch die Schnittstelle nach außen getragen werden sollen, gewahrt.

b) In Java können Interfaces als eigenständiges Konstrukt (Schlüsselwort `interface`) deklariert werden. Welche Zugriffsmodifikatoren dürfen die unmittelbar enthaltenen Deklarationen haben?

**Lösung:**

Ein Interface ist eine öffentliche Schnittstelle. Konsequenterweise können in Interfaces, anders als in Klassen, die unmittelbar enthaltenen Deklarationen keine Zugriffsmodifikatoren außer dem implizit angenommenen `public` haben — dies wäre für eine Schnittstelle auch unsinnig.

## Aufgabe 11 (Polymorphie)

**12 Punkte**

Nennen Sie drei im Kurs genannte Formen von Polymorphie und erklären Sie deren Bedeutung.

**Lösung:**

Im Kurs werden u. a. folgende Arten von Polymorphie beschrieben:

1. *Inklusionspolymorphie:* Programmkonstrukte bzw. Programmteile, die für Objekte eines Typs `T` formuliert sind, sind unverändert auch für Objekte eines Typs `S` verwendbar, wenn `S` ein Subtyp von `T` ist. D. h., Objekte vom Typ `S` können an allen Programmstellen verwendet werden, an denen Objekte vom Typ `T` zulässig sind. Insbesondere können mit Hilfe der Inklusionspolymorphie inhomogene Behälter realisiert werden, d. h. Behälter, die Objekte verschiedener Typen enthalten können.
2. *Parametrische Polymorphie:* Dies ist eine Art der Polymorphie, bei der man Programmkonstrukte oder -teile — in objektorientiertem Kontext sind das dann meist Klassen und Methoden — mit Typen parametrisiert. So kann man z. B. Behältertypen mit ihren Elementtypen parametrisieren. Der einmal für diesen Behälter geschriebene Programmcode kann durch einfaches Instanziieren des Typparameters für jeden beliebigen Elementtyp verwendet werden. Parametrische Polymorphie bietet weniger Flexibilität als Inklusionspolymorphie. Beispielsweise lassen sich mit Sprachen, die nur parametrische Polymorphie unterstützen, keine inhomogenen Behälter realisieren. Andererseits gestattet parametrische Polymorphie bei vielen Anwendungen eine leistungsfähigere Typanalyse zur Übersetzungszeit.

3. *Beschränkt parametrische Polymorphie:* Beschränkt parametrische Polymorphie verbindet Inklusionspolymorphie mit parametrischer Polymorphie. Sie ermöglicht es, Typparameter von parametrischen Typen durch Angabe von Supertypen zu beschränken. Für einen durch einen Typ `T` beschränkten Typparameter `E` dürfen nur solche Typen `S` eingesetzt werden, die Subtypen von `T` sind. Bei der Implementierung solcher beschränkt parametrischer Typen durch Klassen kann dann das Wissen ausgenutzt werden, daß Objekte eines Typs `S`, der für einen Typparameter `E` eingesetzt wird, der selbst nach oben durch einen Supertyp `T` beschränkt ist, mindestens über alle Eigenschaften von `T` verfügen. Diese Eigenschaften können dann in der Implementierung benutzt werden.

## Aufgabe 12 (Probleme der objektorientierten Programmierung)

**4 + 6 Punkte**

a) Skizzieren Sie das Problem der mangelnden Kapselung.

**Lösung:**

Durch das Aliasing-Problem wird die Kapselung von Daten aufgebrochen. Das Aliasing-Problem bezeichnet die Tatsache, dass durch Aliase das Verbergen eines Objekts hinter der Schnittstelle eines anderen Objekts umgangen werden kann. So nützt es z. B. nichts, wenn in Smalltalk die Instanzvariablen eines Objekts von außen nicht zugreifbar sind, wenn das Objekt, auf das eine Instanzvariable verweist, einen Alias außerhalb des Objekts hat: Das Objekt ist damit über den Alias zugreif- und änderbar. Lediglich die Instanzvariable ist geschützt (so dass ihr Inhalt, der Verweis auf das Objekt, nicht geändert werden kann).

b) Nennen Sie die weiteren im Kurs behandelten Probleme der objektorientierten Programmierung.

**Lösung:**

1. Problem der Substituierbarkeit
2. Fragile-base-class-Problem
3. Problem der schlechten Tracebarkeit
4. Problem der eindimensionalen Strukturierung
5. Problem der mangelnden Skalierbarkeit
6. Problem der mangelnden Eignung

## Aufgabe 13 (Richtig oder Falsch)

**10 Punkte (für jede falsche Antwort 1 Minuspunkt!)**

Kreuzen Sie an, ob die folgenden Aussagen richtig oder falsch sind. Für jede richtige Antwort gibt es einen Punkt; für jede falsche Antwort wird ein Punkt abgezogen. Nicht beantwortete Fragen werden mit null Punkten bewertet. Werden bei dieser Aufgabe insgesamt Minuspunkte erreicht, wird diese Aufgabe mit null Punkten bewertet.

| | **Richtig** | **Falsch** |
|---|---|---|
| Zwei Objekte sind identisch, wenn sie nicht zwei Objekte, sondern nur eines sind. | X | |
| Identische Objekte müssen nicht unbedingt gleich sein. | | X |
| Ein Objekt kann zur gleichen Zeit mehrere verschiedene Namen haben. | X | |
| Aliasing führt dazu, dass jedes Objekt nur einmal vorkommt. | | X |
| Lokale Variablen überdecken immer globale Variablen gleichen Namens. | X | |
| Wenn eine Variable nicht sichtbar ist, kann auf das von ihr bezeichnete Objekte nicht zugegriffen werden. | | X |
| Ein Typ hat eine Intension und eine Extension, wobei erstere der Definition des Typs entspricht, letztere seinem Wertebereich. | X | |
| Java erlaubt keine Mehrfachvererbung. | X | |
| Das Friends-Konzept von C++ ermöglicht das gezielte Auffinden von Objekten, die den gleichen Namen haben. | | X |
| Typkonformität ist eine symmetrische Eigenschaft: Wenn ein Typ A typkonform zu einem Typ B ist, dann ist B genauso typkonform zu A. | | X |