Aufgaben

Handout PRG2_OOP1 (Objektorientierte Programmierung)

1. Beantworten Sie die Kontrollfragen A und B, soweit diese im Unterricht noch nicht beantwortet wurden. Halten Sie allfällige Unklarheiten und Fragen schriftlich fest.
   Siehe PRG2_OOP1_KF.docx / PRG2_OOP1_KF.pdf

Handout PRG2_DAT1 (Datenstrukturen)

2. Beantworten Sie die Kontrollfragen A, B, C und D, soweit diese im Unterricht noch nicht beantwortet wurden. Halten Sie allfällige Unklarheiten und Fragen schriftlich fest.
   Siehe PRG2_DAT1_KF.docx / PRG2_DAT1_KF.pdf

3. Analysieren Sie die nachfolgenden Anwendungsszenarien und wählen Sie aus den folgenden vier Datenstrukturen jene aus, die sich am besten eignet. Zusätzlich notieren Sie in einem Satz, was für Ihre Wahl ausschlaggebend war.
   - Array
   - ArrayList
   - SortedArray (wie Array, jedoch sortiert)
   - SortedArrayList (wie ArrayList, jedoch sortiert)

   a) Einer Klasse `IntegerRanking` kann per Methodenaufruf `reportInteger(int i)` ein Integer Wert mitgeteilt werden. Die Klasse speichert jeweils die 100 grössten gemeldeten Integer Werte. Die gespeicherten Werte können per Methodenaufruf `printRanking()` ausgegeben werden.

   b) Bei einer Klasse `ConnectionLogger` kann per Methodenaufruf `reportConnection(String connectionID, Long timestamp)` eine Verbindung registriert werden. Mit Hilfe der Methode `Long[] getConnection(String connectionID)` kann getestet werden, ob eine Verbindung existiert. Falls eine existiert, werden die Zeitstempel dieser und der maximal nächsten neun zeitlich auf diese Verbindung folgenden Verbindungen zurückgegeben. Ansonsten wird `null` zurückgeliefert.

   c) Einer Klasse `IntegerRanking` wird per Methodenaufruf `reportInteger(int i)` ein Integer Wert mitgeteilt. Die Klasse speichert alle gemeldeten Integer Werte und gibt diese bei einem Aufruf der Methode `printRanking()` in sortierter Reihenfolge aus. `printRanking()` wird meistens nur ein einziges Mal aufgerufen.

   a) SortedArray
   feste Grösse, sortiert (die 100 grössten)

   b) SortedArrayList (eigentlich wäre dies doch etwas für HashMap, sofern connectionID eindeutig ist)
   dynamische Grösse, sortiert damit die nachfolgenden 9 Verbindungen ausgegeben werden können

   c) ArrayList
   dynamische Grösse, da printRanking() nur ein einziges Mal aufgerufen wird, kann das Sortieren dann erfolgen und somit ist das einfügen schneller.

Kapitel 8.6 und 8.7 (Repetition)

4. zu bearbeitende Aufgabe:
   - Öffnen Sie das Projekt dome-v2.
   - Studieren Sie den Source-Code der 4 Klassen (`Item`, `CD`, `DVD` und `Database`).
   - Erzeugen Sie ein `Database`-Objekt namens `db`.
   - Erzeugen Sie ein `CD`-Objekt namens `cd1` (Konstruktor mit: Titel, Künstler, Anzahl Stücke, Spieldauer).
   - Erzeugen Sie ein `DVD`-Objekt `dvd1` (Konstruktor mit: Titel, Regisseur/in, Spieldauer).
   - Fügen Sie `cd1` und `dvd1` in Ihre Datenbank ein.
   - Geben Sie mit `list()` den Inhalt Ihrer Datenbank aus.

```
public static void main(String[] args) {
    Database db = new Database();
    CD cd1 = new CD("Weltreise", "Schiller", 18, 72);
    DVD dvd1 = new DVD("Das Boot", "Director", 240);
    db.addItem(cd1);
    db.addItem(dvd1);
    db.list();
}
```

5. Was für Vorteile bringt uns das Konzept der Vererbung?
   – einfache Wiederverwendung (von Implementation)
   – Vermeidung von Code-Duplikation
   – häufig einfachere Wartbarkeit
   – häufig einfachere Erweiterbarkeit

6. Hat die Vererbung auch Nachteile? Denken Sie an die Kopplung.
   Ja, starke Kopplung !

7. Inwiefern wird in obiger Aufgabe bzw. im Source-Code von Substitution Gebrauch gemacht?

```
private ArrayList<Item> items;
public void addItem(Item theItem)
{
    items.add(theItem);
}
```

   Bei addItem() wird ein Subtyp (cd resp. dvd) dem Basis-Objekt (item) zugewiesen.

Kapitel 9.1 und 9.2

8. zu bearbeitende Aufgaben (dome-v2): 9.1

Ein Compilieren ist nicht mehr möglich.

Die Klasse Database verwendet in der Methode list() die Methode print(), welche auf die lokalen Variablen title, playingTime und gotIt sowie comment (alle sind als private deklariert in der Klasse item) zugreifen möchte.

Wenn die Methode print() nun in die Klasse CD resp. DVD verschoben wird ist dies nicht mehr möglich.

9. Vererbung wird mit einer "Einbahnstrasse" verglichen. Weshalb?

Es kann nur von „oben nach unten“ vererbt werden, nicht aber zusätzlich umgekehrt.

Vererbung ist unidirektional.

10. Aufgabe 9.1 (vgl. Verschieben von `print()` in die Unterklassen) führt zu Fehlermeldungen beim Compilieren der Klassen `CD` / `DVD` sowie bei `Database`. Was sind die Ursachen?

Falls in einer Unterklasse die Methode print() nicht implementiert ist, wird diese in der Basisklasse gesucht und falls gefunden ausgeführt (auch über mehrere Vererbungen hinweg).

Da diese nun in eine Unterklasse verschoben wurde, kann sie von den anderen Unterklassen nicht mehr ausgeführt (gefunden) werden.

11. Was versteht man unter dem statischen Typ?

```
Konto k;
```

Die Referenzvariable k besitzt den statischen Datentyp Konto.

12. Was versteht man unter dem dynamische Typ? Was ist dabei dynamisch?

```
k = new Konto();
k = new Spar();
k = new Giro();
```

k besitzt nacheinander die dynamischen Datentypen Konto, Spar und Giro.

Dynamisch bedeutet auf Ebene Ausführung bzw. zur Laufzeit.

13. Müssen statischer Typ und dynamischer Typ übereinstimmen? Falls nein, was für "Spielregeln" gelten?

Nein, eine Referenzvariable vom Typ einer Unterklasse kann auch auf ein Objekt der dazugehörigen Basisklasse zeigen.

14. Überprüft der Compiler statische oder dynamische Typen?

statische Typen

Kapitel 9.3 und 9.4

15. zu bearbeitende Aufgabe (dome-v3): unten beschriebene Experimente
?

16. Was bedeutet Überschreiben?
`@Override`
In einer Unterklasse wird eine Methode aus der Basisklasse mit gleichem Name und Signatur erstellt (überschrieben).
Beispiel:

```
public class Konto
{
    ...

    public void print()  // allg. Implementation in der Basisklasse
    {
        System.out.println("Saldo: " + saldo);
    }
    ...
}
```

```
public class Giro extends Konto
{
    ...

    @Override
    public void print()  // in Unterklasse spezifisch implementiert!
    {
        System.out.println("Kreditlimite: " + creditLimit);
    }
    ...
}
```

17. Die Signatur ist im Zusammenhang mit dem Überschreiben wichtig. Wie ist die Signatur einer Methode definiert? Machen Sie hierfür folgende zwei Experimente: Ändern Sie in der Klasse `CD` den Rückgabewert der Methode `print()` von `void` auf `int` ab, compilieren Sie dann die Klasse; ändern Sie den Zugriffsmodifizierer von `public` auf `private`, compilieren Sie erneut.
Signatur: Enthält den Methodennamen und die Parameterliste (nicht aber den Rückgabewert).
Nur wenn die Signatur, Rückgabewert und Zugriffsmodifizierer übereinstimmen ist ein @Override möglich.

```
CD.java:46: print() in CD cannot override print() in Item; attempting to use incompatible return type
found   : int
required: void
    public int print()

CD.java:46: print() in CD cannot override print() in Item; attempting to assign weaker access privileges;
was public
    private void print()
```

18. Ist der statische oder der dynamische Typ beim Aufrufen einer Methode massgebend?
Der dynamische Typ ist massgebend. (Aufruf einer Methode ist zur Laufzeit)

19. Auf den Seiten 265 und 266 wird sehr schön das method lookup (method binding, method dispatching) beschrieben und illustriert. Welche Methode hat bei überschriebenen Methoden den Vorrang, jene in der Oberklasse oder jene in der Unterklasse?
Es wird automatisch die „naheliegendste“ Methode aufgerufen. Somit also diejenige der Unterklasse.
Wird in dieser Klasse keine Methode gefunden, so wird die Basisklasse „nach oben“ nach der Methode durchsucht.

Kapitel 9.5 und 9.6

20. zu bearbeitende Aufgabe (dome-v3): 9.3 und 9.4

9.3:

```
Weltreise (72 mins)
    <no comment>
    Schiller
    tracks: 18
Das Boot (240 mins)
    <no comment>
    director: Director
```

Es funktioniert wie erwartet, jedoch kann die Reihenfolge der Ausgabe dadurch nicht beeinflusst werden.

9.4:

Bsp. CD: System.out.print("CD: ") hinzufügen.

```
    public void print()
    {
        System.out.print("CD: ");
        super.print();
        System.out.println("    " + artist);
        System.out.println("    tracks: " + numberOfTracks);
    }
```

21. Was bewirkt die Anweisung `super()`? Wo muss diese Anweisung stehen?

Es wird der Konstruktor der Basisklasse aufgerufen.

Sie muss an erster Stelle im Konstruktor der Unterklasse stehen.

22. Was bewirkt die Anweisung `super.print()`? Wo darf diese Anweisung stehen?

Sie ruft die Methode print() der Basisklasse auf.

Überall in einer Unterklasse von der Basisklasse wo diese Methode implementiert ist.

23. Polymorphie übersetzen wir mit Vielgestaltigkeit. Inwiefern kommt Polymorphie beim Überladen und beim Überschreiben von Methoden zum Ausdruck?
(Bemerkung: Im Zusammenhang mit Überladen spricht man häufig von schwacher Polymorphie und beim Überschreiben von starker Polymorphie.)

Beim Überladen / Überschreiben wird immer der gleiche Methodennamen verwendet, jedoch wird diese Methode unterschiedlich implementiert.

Kapitel 9.7 und 9.8

24. zu bearbeitende Aufgabe (dome-v3): 9.5 und 9.6

9.5:

Die Methode toString() erwartet keine Parameter. Rückgabewert ist String.

9.6:

```
toString(): DVD@9304b1
```

25. Die Default-Implementation von `toString()` in der Klasse `Object` führt zu einer speziellen Ausgabe. Im Buch ist von "magic number" die Rede. Was steckt dahinter?

[Klassenname]@[Adresse auf dem Heap]

Die „Magic Number" ist die Adresse im Heap wo dieses Objekt gespeichert ist.

26. Wieso überschreibt man häufig `toString()`?

Die Standard Ausgabe macht oft wenig Sinn.

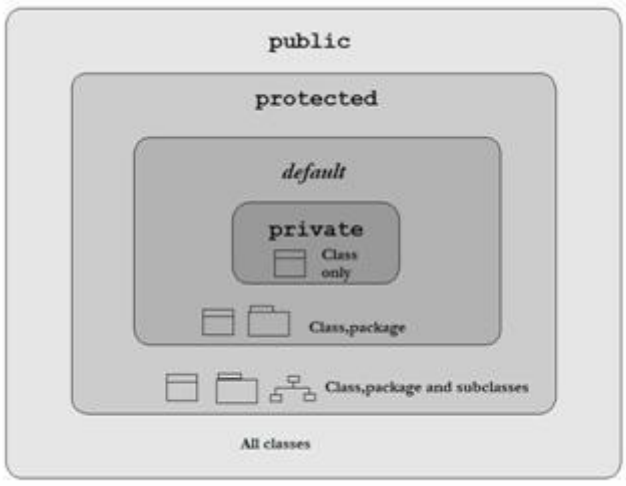
27. Wie verhalten sich die altbekannten Methoden `System.out.println()` und `System.out.print()`, falls man ihnen als Parameter keine Strings übergibt?

Wenn kein String als Parameter übergeben wird, wird die toString() Methode des übergebenen Objektes aufgerufen.

28. Inwiefern unterscheiden sich die Zugriffsmodifizierer `private`, `protected` und `public`? (Der Zugriffsmodifizierer `protected` wird im Lehrbuch nicht ganz präzise beschrieben! Lesen Sie doch mal im Softbook von "Krüger" nach (www.javabuch.de, Index `protected`).)

Zugriff von: Modifizierer:	eigener Klasse	Klasse in gleichem Package	Unterklasse in anderem Package	Nicht Unter-klasse in anderem Package
`private`	ja	nein	nein	nein
- Standard bzw. Package Scope	ja	ja	nein	nein
`protected`	ja	ja	ja*	nein
`public`	ja	ja	ja*	ja*

(*) Die Klasse muss dann natürlich importiert sein.



Quelle: http://www.noesispoint.com/jsp/scjp/SCJPch3.htm