Programmieren 2
Selbststudium Semesterwoche 1

Kapitel 8.6 und 8.7
4. Was für Vorteile bringt uns das Konzept der Vererbung?
→ Code Duplikationen können weitgehend vermieden werden.
→ Sub-Klassen enthalten nur spezialisierenden Code, welcher nicht schon in der super-Klasse geschrieben wurde.
→ Sub-Klassen dürfen auch neue Methoden und Attribute enthalten.

5. Hat die Vererbung auch Nachteile? Denken Sie an die Kopplung.
→ Ja, Vererbung bringt auch Nachteile mit sich (auch wenn diese meist vernachlässigbar sind): Durch die Vererbung von Attributen/Methoden entstehen Abhängigkeiten zwischen den Klassen. Diese Abhängigkeiten erhöhen die Kopplung, was die Komplexität erhöhen und die damit verbundene Entwicklungsfreundlichkeit (Stichwort: Wartbarkeit, Erweiterbarkeit) verschlechtern kann.

6. Inwiefern wird in obiger Aufgabe bzw. im Source-Code von Substitution Gebrauch gemacht?
→ In der Methode „addItem“ wird einem Item Objekt implizit ein CD bzw. DVD Objekt zugewiesen:
→ Item i = new CD(…)
→ Siehe Folie 13/14 in PRG2_SW1_OOP.pdf.

Kapitel 9.1 und 9.2
8. Vererbung wird mit einer "Einbahnstrasse" verglichen. Weshalb?
→ Eine Vererbung ist unidirektional, d.h. sie hat nur Auswirkungen in eine Richtung.
(CD cd1 = new Item(…) würde einen Compilerfehler verursachen)
→ Eine Vererbung kann auch in keinem Fall nicht horizontal erfolgen.
(CD cd1 = new DVD(…) würde ebenfalls einen Compilerfehler verursachen)
→ Siehe Folie 13/14 in PRG2_SW1_OOP.pdf.

9. 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?
→ Die Klasse Database den Ausdruck item.print(), welcher eine Abhängigkeit zur Klasse Item darstellt. Wir die Print-Funktion verschobe, muss auch der Code in der Klasse Database angepasst werden.
→ Wird die Print-Funktion in die Klassen CD bzw. DVD verschoben, so muss deren Code ebenfalls angepasst werden. (Zugriff auf lokale Attribute ist nicht mehr in jedem Fall gewährleistet).

10. Was versteht man unter dem statischen Typ?
→ Item i = new Item(…); Die Referenzvariable i besitzt den statischen Datentyp Item. Statisch bedeutet auf Ebene Source-Code bzw. zur Compilationszeit.

11. Was versteht man unter dem dynamische Typ? Was ist dabei dynamisch?
→ Während der Laufzeit des Programms kann „Item i“ einen anderen, dynamischen Datentyp annehmen. Beispielsweise i = new CD(…).

12. Müssen statischer Typ und dynamischer Typ übereinstimmen? Falls nein, was für "Spielregeln" gelten?
→ Nein. Ein dynamischer Typ muss aber stets eine Erbung eines statischen Typs sein.

13. Überprüft der Compiler statische oder dynamische Typen?
→ Nur statische Typen!

Kapitel 9.3 und 9.4

15. Was bedeutet Überschreiben?

→ Überschreiben (Java: „Overriding“) bedeutet, dass eine Vererbung (Attribut/Methode) in der Sub-Klasse neu implementiert wird.

→ Beispiel:

```
public class Konto
{
    ...

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

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

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

16. 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.

→ Die Signatur einer Methode enthält den Methodennamen und die Parameterliste.

→ Versuch 1: Die Signatur der Methode print() der Klasse CD stimmt nicht mehr mit der Signatur der Methode print() der Klasse Item überein.

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

```
print() in CD cannot override print() in Item; attempting to use incompatible re
turn type
```

→ Versuch 2: Die Einschränkung des Zugriffs auf print() der Klasse CD ist nicht möglich.

```
    private int print()
    {
        System.out.println("    " + artist);
        System.out.println("    tracks: " + numberOfTracks);
    }
}
```

```
print() in CD cannot override print() in Item; attempting to assign weaker acces
s privileges; was public
```

17. Ist der statische oder der dynamische Typ beim Aufrufen einer Methode massgebend?

→ Der dynamische Typ ist massgebend.

18. 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?

→ Wird eine aufgerufene Funktion in derselben Klasse nicht gefunden, so wird die super-Klasse nach der Methode durchsucht. Dies geschieht Schritt für Schritt entlang der Vererbungshierarchie bis zur Klasse Object.

Kapitel 9.5 und 9.6

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

→ Das Schlüsselwort super ermöglicht den Aufruf von Methoden in super-Klassen.

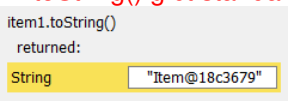
21. Was bewirkt die Anweisung super.print()? Wo darf diese Anweisung stehen?

→ Die Anweisung ruft die print()-Methode der übergeordneten Klasse auf.

→ super.print() darf nur dort aufgerufen werden, wo eine vererbte, übergeordnete Klasse eine öffentliche print()-Methode enthält.

22. 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 Überschreiben/Überladen von Methoden benutzen wir ein und denselben Namen für unterschiedliche Implementationen.

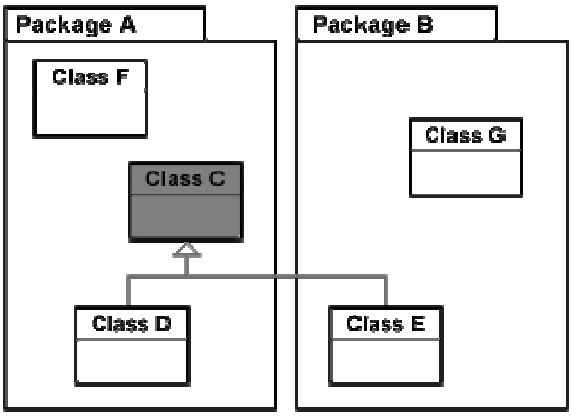
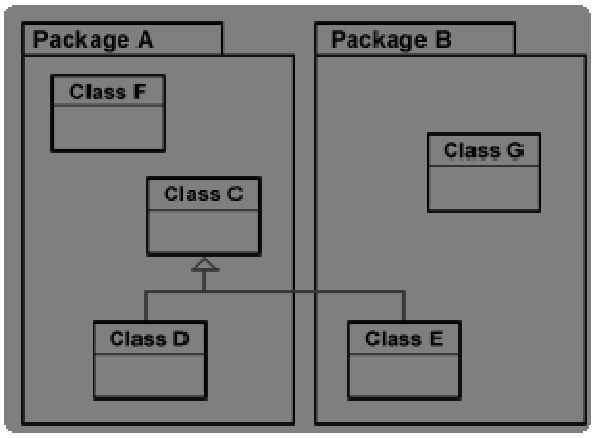
Kapitel 9.7 und 9.8
24. Die Default-Implementation von toString() in der Klasse Object führt zu einer speziellen Ausgabe. Im Buch ist von "magic number" die Rede. Was steckt dahinter?
→ toString() gibt standardmässig <Klassenname>@<Referenz-ID> zurück.

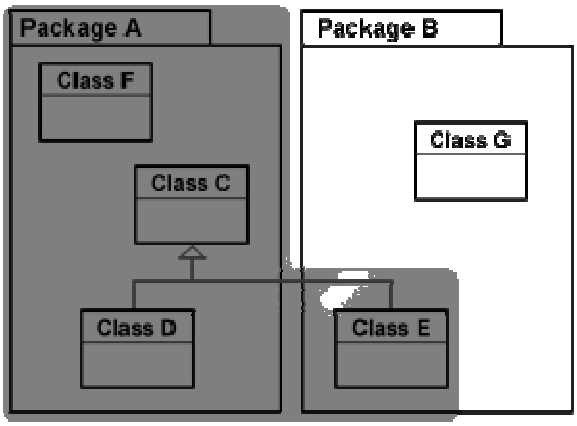
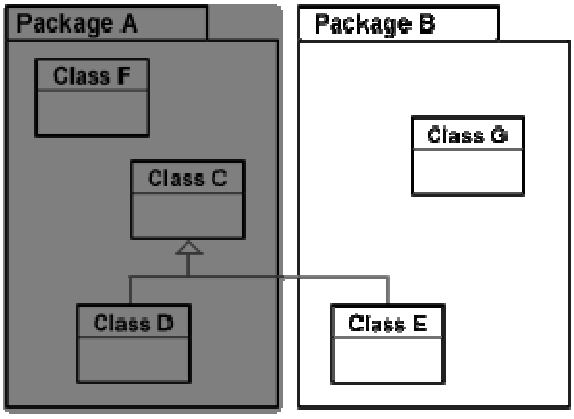


25. Wieso überschreibt man häufig toString()?
→ Weil die standardmässige toString() Methode nicht viel Sinn macht.

26. Wie verhalten sich die altbekannten Methoden System.out.println() und System.out.print(), falls man ihnen als Parameter keine Strings (d.h. andere Typen/Objekte) übergibt?
→ Wenn keine Strings übergeben werden, wird die toString() Methode ausgeführt. (Siehe Aufgabe 24).
→ Beispiel: System.out.println(new Integer(100));

27. 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).)
→ Folgende Tabelle illustriert die Zugriffscharakteristik der einzelnen Zugriffsmodifizierer in Java:

`private`	
`public`	

protected	
no acces modifier	

Quelle: http://www.emf4net.org/Translation/AccessModifiers/tabid/77/Default.aspx