11. Übung am 17.01.2011
==========

Anleitung:

Während jeder Übung ist von jeder Gruppe ein (kurzes) Protokoll zu erstellen. Das Protokoll ist eine einfache ASCII-Text-Datei, die mit einem Text-Editor (nedit, kedit, kate) mit dem Sie auch Ihre Programme schreiben, erzeugt wird. Nennen Sie diese Datei unbedingt "PROTOKOLL.txt".

Das Protokoll muss Folgendes enthalten:
1. Datum, Übungsnummer, Gruppennummer, Name(n) der mitwirkenden Studierenden,
2. Benötigter Zeitaufwand für die gestellten Aufgaben (circa),
3. Namen der erstellten Programme (KEINE Listings),
4. Kurze Antwort auf eventuell weiter unten gestellte Fragen,
5. Eventuelle Probleme oder Besonderheiten, falls diese aufgetreten sind.

Sämtliche während der Übung erstellten Dateien (Protokoll, Source Codes, ausführbare Programme, etc.) verbleiben im Verzeichnis für den jeweiligen Übungstag, also z.B. "11Ue2011-01-17/" Ihrer Gruppe.

Das Protokoll und die Übungsprogramme sollten am jeweiligen Übungsnachmittag erstellt werden, spätestens jedoch bis zum nächstfolgenden Übungstag (Montag), 14:00! (Spätere Ausarbeitungen können nur in begründeten Fällen und nach Rücksprache mit Ihrem Betreuer bzw. Tutor berücksichtigt werden!)

==========

Aufgabe zu Kapitel 8 (3 Punkte):

Schreiben Sie eine Klasse **person** mit den zwei Attributen **nachName** und **vorName**, sowie eine Klasse **student** und eine Klasse **tutor**, die beide von **person** erben. Die Klasse **student** hat ein Attribut **matrikelnummer**, die Klasse **tutor** ein Attribut **gebiet**. Der Einfachheit halber seien alle Attribute vom Typ *string*. Fügen Sie Methoden zum Lesen der Attribute in den jeweiligen Klassen hinzu wie zum Beispiel *const string& get_nachName()* in der Klasse **person**. Es soll auch eine Methode *to_String()* geben, welche die vollständigen Informationen liefert, und deren Schnittstelle und eine Standard-implementierung in der Klasse **person** definiert ist. Letztere soll einen aus Vor- und Nachnamen zusammengesetzten String zurückliefern die in den Unterklassen zu redefinierenden Implementierungen auch noch den Status (Student/Tutor) und die Matrikelnummer bzw. das Betreuungsgebiet. *Achtung:* Es gibt also verschiedene „Versionen" von *to_String()*, d.h. diese Methode sollte ..... implementiert werden. Von der Klasse **person** soll kein Objekt erzeugt werden können, sie sei also abstrakt. Der folgende Programmauszug zeigt die Benutzung der Klassen:

```
vector<person*> diePersonen;
diePersonen.push_back(
    new student("Mueller", "Antonia","200983"));
diePersonen.push_back(
    new tutor("Gidaly", "Gottfried", "Magnetismus"));
diePersonen.push_back(
    new student("Kuhn", "Leopold", "223446"));

cout << "Vornamen: " << endl;
for(size_t i = 0; i < diePersonen.size(); ++i) {
   cout << diePersonen[i]->get_vorName() << endl;
}

cout << "Nachnamen: " << endl;
for(size_t i = 0; i < diePersonen.size(); ++i) {
   cout << diePersonen[i]->get_nachName() << endl;
}

cout << "to_String() ergibt: " << endl;
for(size_t i = 0; i < diePersonen.size(); ++i) {
   cout << diePersonen[i]->to_String() << endl;
}
```

Die Ausgabe des obigen Programms lautet:

Vornamen:
Antonia
Gottfried
Leopold
Nachnamen:
Mueller
Gidaly
Kuhn
to_String() ergibt:
Student/in Antonia Mueller, Mat.Nr.: 200983
Tutor/in Gottfried Gidaly, Gebiet: Magnetismus
Student/in Leopold Kuhn, Mat.Nr.: 223446