Klausur Problemorientiertes Programmieren (PPmP) SS05 - Döben-Henisch

7.Juli 2005, 10:00h – 12:00h, Raum BCN103-105

NAME	
MATRIKELNUMMER	
Prüfungsart:	

Summe Pkte		**Note**	

Minimale Punktezahl, um PPmP zu bestehen: 50 Pkte (SL: 25 Pkt Klausur + 25 Pkt Übungen)

1. Regularien

i. Teilnehmer an der Klausur müssen sich durch Lichtbildausweis identifizieren.
ii. Die Klausur findet am PC statt. Bücher und Skripte sind als Hilfsmittel zugelassen.
iii. Die Klausurzeit beträgt 120 Min.
iv. Alle Ergebnisse muss man in eine dafuer vorgesehene Datei abspeichern, die die eigene Matrikelnummer im Dateinamen enthaelt. Alle abgespeicherten Dateien muessen zusaetzlich Namen und Matrikelnummer enthalten. Vor Verlassen des Raumes muss sich jeder vergewissern, ob die Dateien korrekt abgespeichert worden sind.
v. Es gilt folgende Punktetabelle:

NOTE	PUNKTE
1	ab 85
2	75 bis < 85
3	65 bis < 75
4	50 bis < 65
5	<50

vi. Das aktive Abschreiben wie auch das Zulassen von Abschreiben wird als Täuschungsversuch bewertet.
vii. Jeder, der an der Klausur teilnimmt, erklärt zu Beginn, dass er sich durch keine Umtände beeinträchtigt empfindet, die seine Klausurleistung wesentlich beeinflussen könnten.
viii. Auf dem Deckblatt sind alle Dateien einzutragen, die man abgespeichert hat:

Dateien: .....
Dateien:.....
Dateien:.....
Dateien:.....
Dateien:.....
Dateien:.....
Dateien:.....
Dateien:.....
Dateien:.....

Aufgabe 1 (Max. 30 Pkte).

Es soll ein Programm für einen Getränkeautomat geschrieben werden. Es wird hier nur die Sicht des Benutzers berücksichtigt. Der Benutzer kann aus den vorhandenen Getränkesorten eine bestimmte Sorte auszuwählen. Es sollen drei Sorten verfügbar sein. Der Benutzer gibt sein Geld ein. Der eingegebene Betrag muss mit dem geforderten Betrag übereinstimmen (Geld muss passend sein; kein Wechselgeld). Falls das Geld nicht korrekt ist, wird es zurückgegeben. Ist das Geld korrekt, dann wird eine Flasche der gewünschten Sorte ausgegeben. In beiden Fällen geht der Automat wieder in seine Wartestellung zurück. Sind nicht mehr genug Flaschen da, schaltet sich der Automat ab.

(i) (10 Pkte) Erstellen Sie mit umbrello ein Anwendungsfalldiagramm (Use Case)
(iii) (20 Pkte) Erstellen Sie mit umbrello ein Klassendiagramm (Attribute + Methoden)

Hinweis: Legen sie mit umbrello 1 Datei mit Namen *Usecase* neu an (*Speichern als*) und innerhalb dieser einen Datei erzeugen sie das Anwendungsfalldiagramm und das Klassendiagramm. Sichern Sie mit *Ctrl-S (Speichern)* immer wieder den aktuellen Stand.

Aufgabe 2 (Max. 65 Pkte):

Erstellen Sie auf der Grundlage von Aufgabe A1 eine Headerdatei (*automat.hpp*), eine Implementierungsdatei (*automat.cpp*) sowie eine Usagedatei (*automat_usage.cpp*). Benutzen Sie für die Usage-Datei das folgende Schema als Ausgangspunkt (siehe Ordner VORLAGEN):

```
//---------------------------------------
// automat_usage.cpp
//
// Klasse fuer einen Getraenkeautomaten mit Name Automat.
// Es gibt drei Sorten von Getraenken.
// Zu Beginn hat jede Sorte 3 Flaschen
// Bei Ausgabe einer Flasche vermindert sich der Speicher um 1
// Wenn der Speicher von allen Flaschen leer ist, dann stoppt der Automat.
//
// Compilation: g++ -o automat automat.cpp automat_usage.cpp
//
// Usage: ./automat
//---------------------------------------

#include "automat.hpp"
#include <iostream>

using  namespace std;

int main(){

  // (10 Pkte) Starte Objekt Automat mit 3 Sorten und 3 Flaschen pro Sorte

  // (10 Pkte) Anzeige aktuell verfuegbare Sorten (max.3)

  // (10 Pkte) Auswahl einer Sorte
```

```
  // (10 Pkte) Anzeige Preis

  // (10 Pkte) Eingabe Geld

  // (10 Pkte) Falls Geld falsch, dann Rueckgabe und Start von vorne
  //Falls Geld OK, dann Ausgabe Flasche und  Start von vorne

  // (5 Pkte) Wenn alle Flaschen verbraucht, dann Stopp

}
```

Aufgabe 3 (Max. 40 Pkte):
Gegeben sind drei Dateien: *stack.hpp, stack.cpp* und *stack_usage.cpp* (siehe Ordner VORLAGEN). Folgende Aufgaben können Sie unabhängig voneinander lösen:

(i) (20 Pkte) Verändern Sie die Dateien so, dass aus der Klasse Stack ein Template wird. Dazu generieren Sie die Dateien *stack_template.hpp, stack_template.cpp* und *stack_template_usage.cpp.*
(ii) (20) Verändern sie den Text so, dass es eine Ausnahmebehandlung mit try, throw und catch so gibt, dass der Fehler durch Zugriff auf leeren Stack abgefangen wird. Dazu generieren Sie die Dateien *stack_err.hpp, stack_err.cpp* und *stack_err_usage.cpp (Hinweis: Die Klasse Vector hat eine Methode empty(), die true liefert, wenn der Vector leer ist).*

```
//--------------------------------
// stack.hpp
//--------------------------------

#include <vector>
#include <string>

using namespace std;

class Stack {
  private:
    vector<string> elems;  // Elemente

  public:
    Stack();                  // Konstruktor
    void push(const string&);   // Element einkellern
    string pop();                 // Element auskellern
    string top() const;             // oberstes Element

};

//------------------------
// stack.cpp
//-------------------------

#include "stack.hpp"
```

```
using namespace std;

// Konstruktor
Stack::Stack ()
{
   // nichts mehr zu tun
}

void Stack::push (const string& elem)
{
   elems.push_back(elem);    // Kopie einkellern
}

string Stack::pop ()
{
   string elem = elems.back();    // oberstes Element merken
   elems.pop_back();         // oberstes Element auskellern
   return elem;              // gemerktes oberstes Element zurückliefern
}

string Stack::top () const
{
   return elems.back();      // oberstes Element als Kopie zurückliefern
}

//************************************************
// stack_usage.cpp
//****************************************************************
// Compilation:  g++ -o stack_usage stack_usage.cpp stack.cpp
//
// Usage: ./stack1_usage
//*********************************

#include "stack.hpp"
#include <iostream>

using namespace std;

int main()
{
      Stack        stringStack;

       // String-Stack manipulieren
       string s = "hallo";
       stringStack.push(s);
       cout << stringStack.pop() << endl;
       cout << stringStack.pop() << endl; //Fehler, da leer
}
```