Modulprüfung 226

Pflichtenheft „timeManager“

Modulprüfung 226

Pflichtenheft „timeManager“

Version 2010

Technische Berufsschule Zürich
Ausstellungsstrasse 70
8000 Zürich

Autoren:

Informatiker Applikationsentwicklung

Inhaltsverzeichnis

1 Auftrag

Es soll eine detaillierte Arbeitszeiten-Erfassung entwickelt und getestet werden. Folgende Punkte sind sicher zu beachten:

- Die Person, welche ihre Arbeitszeit erfassen will, startet die Anwendung und identifiziert sich mit ihrem Namen.
- Nach der Anmeldung kann die Person einen "Stempel" auslösen. Falls der letzte in der Anwendung erfasste Stempel ein "Anfangsstempel" zu einer Tätigkeit war, wird automatisch ein "Schlussstempel" zu dieser Tätigkeit erfasst. Die Tätigkeit gilt dann als abgeschlossen.

- Falls der letzte in der Anwendung erfasste Stempel aber ein "Schlussstempel" war, muss die Person zuerst die Art der neu zu beginnenden Tätigkeit wählen. Anschliessend kann sie einen "Anfangsstempel" zu dieser Tätigkeit setzen.
- Am Ende des Arbeitstags sollte die Person nicht vergessen, einen Schlussstempel zu setzen.
- Die Anwendung soll aber auf vergessene Schlussstempel reagieren können.
- Leistungsfähige Auswertungen der erfassten Daten sollten allenfalls in verschiedenen Modi möglich sein.

1.1 Weitere Anforderungen:

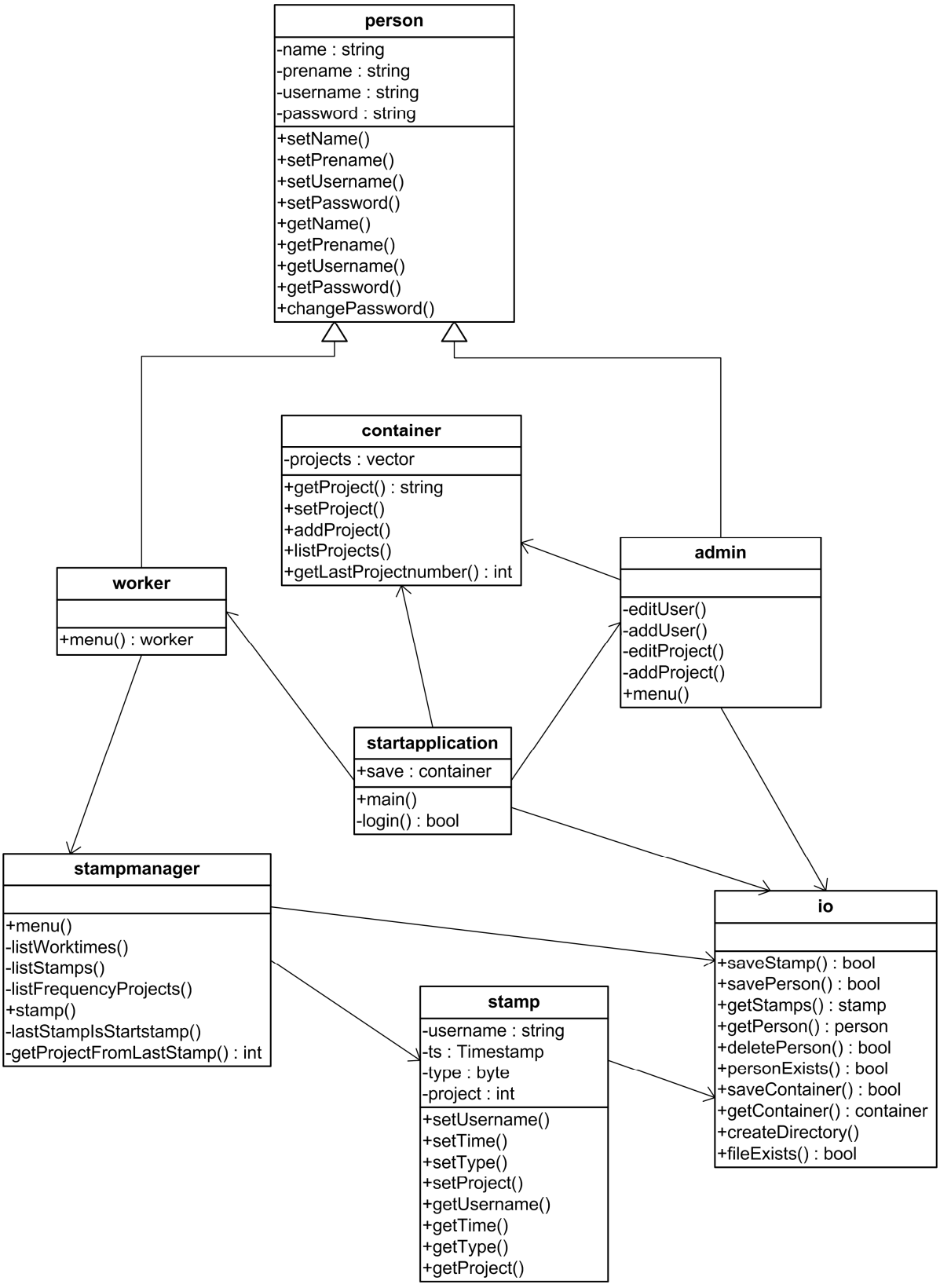
- Es wird davon ausgegangen, dass die Applikation in verschiedenen Umgebungen eingesetzt werden sollte. Daher wird Java als Programmiersprache für die Umsetzung festgelegt.
- Die Anwendung soll die erfassten Zeitdaten in geeigneter Weise speichern.
- Zum Aufbau der Benutzerschnittstelle kann das Package java.awt verwendet werden. Es würde jedoch auch eine zeilenorientierte Benutzerschnittstelle genügen.
- Möglichst benutzerfreundliche Arbeitsweise der Anwendung: Möglichst wenige Eingaben vom Anwender verlangen.
- Möglichst "fehlertolerante" Arbeitsweise der Anwendung: Fehlerfreies Erkennen von Start- und Endstempel und einfache Korrekturmöglichkeit, falls am Ende des Arbeitstags der Endstempel vergessen wurde.

2 Zeitplan

Datum	Peter Muster	Tobias Muster
Meilenstein1: Pflichtenheft und Konzept ausarbeiten		
17.11.2009	Pflichtenheft (Konzept, Idee ausarbeiten / UML beginnen / Zeitplan)	
24.11.2009	Pflichtenheft (UML fertig / Beschreibungen)	
Meilenstein2: Design umsetzen, Quellcode ausarbeiten		
01.12.2009	Klasse „stampmanager“	Klasse „person“
08.12.2009	Klasse “stampmanager”	Klasse “worker”
15.12.2009	Klasse “stamp”	Klasse “admin”
05.01.2010	Klasse “io”	Klasse “admin”
12.01.2010	Klasse „startapplication“	Klasse „container“
Meilenstein 3: Implementation testen (Testsuite)		
12.01.2010	JUnits / Dokumentation JUnits	JUnits / Dokumentation JUnits
Meilenstein 4: Funktionstest		
19.01.2010	Funktionstest	
26.01.2010	Dokumentation Funktionstest	
Präsentation		
02.02.2010	Präsentation Modulprüfung	

3 Design

3.1 UML



3.2 Wichtige Merkmale zu den Klassen

3.2.1 StartApplication

Beim Programmstart wird die Methode „main()“ aufgerufen und das Objekt „save“ erzeugt (von der Festplatte geladen).
Die Methode Login() soll eine Benutzereingabe entgegennehmen. Diese Eingabe umfasst den Benutzernamen und das Passwort. Nach der Eingabe wird das dem Benutzernamen entsprechende Objekt geladen und das Passwort verglichen. Der Methodenkopf soll folgendermassen aussehen:
private static boolean login ()

3.2.2 Person

Diese Klasse enthält eine Methode für die Änderung des Passwortes, sowie set- und get-Methoden für alle Attribute dieser Klasse.
Die Methodenköpfe der bereits erwähnten set- und get-Methoden sollen wie folgt aussehen:
public void setName(String name)
public void setPrename(String prename)
public void setUsername(String username)
public void setPassword(String password)
public String getName()
public String getPrename()
public String getUsername()
public String getPassword()

Die Methode “changePassword” fordert den Benutzer auf, ein neues Passwort einzugeben und überschreibt anschliessend das Attribut „password“ mit dem eingegebenen Wert. Der Methodenkopf dazu soll so aussehen: **public void changePassword()**

3.2.3 Worker

Diese Klasse wird von der Klasse „person“ vererbt. Sie enthält lediglich die Methode „menu()“. Dieses Menu muss so aussehen:

Arbeitermenu
****
1. Stempeln
2. Auswertungen
3. Passwort ändern
4. Abmelden

Bei „Stempeln“ wird die Methode „stamp()“ der Klasse „stampmanager“ ausgeführt.
Bei „Auswertungen“ wird die Methode „menu()“ der Klasse „stampmanager“ ausgeführt.
Bei „Passwort ändern“ wird der Benutzer aufgefordert, sein altes Passwort und danach zweimal das neue Passwort einzugeben. Danach wird es geändert.
Bei „Abmelden“ soll die Wiederholschleife des Menus verlassen werden.

Für diese Klasse muss folgender Konstruktor implementiert werden:
public worker (String name, String prename, String username, String password)

3.2.4 Admin

Diese Klasse wird von der Klasse „person“ vererbt. Sie enthält nebst der Methode „menu()“ noch 5 andere private-Methoden zur Administration. Das Menu muss so aussehen:

Adminmenu
****
1. Projekt editieren
2. Projekt hinzufügen
3. Benutzer erfassen
4. Benutzer löschen
5. Abmelden

Punkt 1 – 4 werden bei den Methoden-Erklärungen näher erläutert. Bei Punkt 5 (Abmelden) soll die Wiederholschleife des Menus verlassen werden.

Die Methode „editProject()“ listet alle Projekte mit Nummer auf und fordert den Admin auf, die Nummer des zu editierenden Projekts anzugeben. Danach soll ein neuer Name für dieses Projekt definiert werden. Um das Projekt zu editieren, wird die Methode „setProject(int i, String project)“ des Objektes „save“ in der Klasse „startapplication“ aufgerufen. Der Methodenkopf sieht so aus:
private void editProject();

Die Methode „addProject()“ fordert den Administrator auf, ein Name für ein neues Projekt einzugeben. Um das Projekt dann zu speichern, wird die Methode „addProject(String project)“ des Objektes „save“ in der Klasse „startapplication“ aufgerufen. Der Methodenkopf sieht so aus:
private void addProject();

Die Methode „addUser()“ fordert den Administrator auf, neue Daten für einen Benutzer einzugeben. Danach wird ein worker-Objekt mit dem Konstruktor, der alle Informationen annimmt erstellt und mit der Methode „savePerson(person p)“ der Klasse „io“ gespeichert. Der Methodenkopf sieht so aus:
private void addUser();

Die Methode „deleteUser()“ fordert den Administrator auf, den Benutzernamen der zu löschenden Person einzugeben. Danach wird das serialisierte Objekt des gelöscht. Der Methodenkopf sieht so aus:
Private void deleteUser();

Für diese Klasse muss folgender Konstruktor implementiert werden:
public admin (String name, String prename, String username, String password)

3.2.5 Stampmanager

Der Stampmanager beinhaltet ein Menu, 3 Auswertungsarten, sowie die Möglichkeit zu stempeln.
Dem Menu muss der Benutzername übergeben werden. Der Methodenkopf sieht daher so aus: **public static void menu(String benutzername)**
Das Menu muss folgendermassen aussehen:

Stampmanager
****
1. Arbeitszeiten auflisten
2. Stempel auflisten
3. Häufigkeit der bearbeiteten Projekte
4. Zurück

Punkt 1 – 3 werden bei den Methoden-Erklärungen näher erläutert. Punkt 4 (Zurück) beendet die Wiederholschleife des Stampmanager-Menus.

Die Methode „listWorktimes()“ listet die Dauer zwischen zwei Stempeln (Anfang- und Endstempel) auf. Dabei werden alle Stempel der entsprechenden Person geladen. Der Methodenkopf soll so aussehen:
private static void listWorktimes(String benutzername)

Die Methode „listStamps()“ listet die Zeiten und Projekte aller Anfangs- und Endstempel auf. Der Methodenkopf sieht so aus:
private static void listStamps(String benutzername)

Die Methode „listFrequencyProjects()“ listet alle Projekte auf, an denen gearbeitet wurde. Es wird ausgerechnet, wie oft an jedem Projekt gearbeitet wurde. Der Methodenkopf sieht so aus:
private static void listFrequencyProjects(String benutzername)

Die Methode „stamp()“ überprüft, ob ein Start- oder End-Stempel erzeugt werden soll. Bei einem Start-Stempel wird zusätzlich die Projektnummer abgefragt. Danach wird mit dem Stempel-Konstruktor ein neues Stempel-Objekt erzeugt. Dabei wird ein Timestamp der aktuellen Systemzeit erzeugt und dem Konstruktor mitgegeben. Der Methodenkopf sieht so aus:
public static void stamp(String username)

Die Methode „getProjectFromLastStamp()“ gibt das Projekt des zuletzt erstellten Stamps einer Person zurück. Der Methodenkopf sieht so aus:
private static int getProjectFromLastStamp(String username)

Die Methode „private static boolean lastStampIsStartstamp()” überprüft, ob der zuletzt erstellte Stamp einer Person ein Startstamp ist. Der Methodenkopf sieht so aus:
private static boolean lastStampIsStartstamp(String username)

3.2.6 Stamp

Diese Klasse enthält keine speziellen Methoden. Dafür muss für jedes Attribut eine set- und eine get-Methode geschrieben werden, die so aussehen sollen:
public void setUsername(String username)
public void setTime(Timestamp ts)
public void setType(byte type)
public void setProject(int project)
public String getUsername()
public Timestamp getTime()
public byte getType()
public int getProject()

Für diese Klasse muss folgender Konstruktor implementiert werden:
Public stamp (String username, Timestamp ts, byte type, int project)

3.2.7 Container

Diese Klasse dient als Speicher. Momentan werden hier die Projekte in einem Vektor gespeichert.
Die Methode „getProject()“ nimmt eine Zahl entgegen, die den Index eines Elements im Vektor „projects“ angibt. Wenn das Element gefunden wurde, wird der Projektname als String zurückgegeben. Der Methodenkopf sieht so aus: **public static String getProject (int i)**

Die Methode „setProject()“ nimmt eine Zahl, die den Index eines Elements im Vektor „projects“ angibt und einen string entgegen. Wenn das Element gefunden wurde, wird der Projektname nach dem übergebenen String geändert. Der Methodenkopf sieht so aus:
public static void setProject (int i, String project)

Die Methode „addProject()“ nimmt einen String entgegen und fügt ihn dem Vektor „projects“ an. Der Methodenkopf sieht so aus: **public static void addProject (String project)**

3.2.8 Io

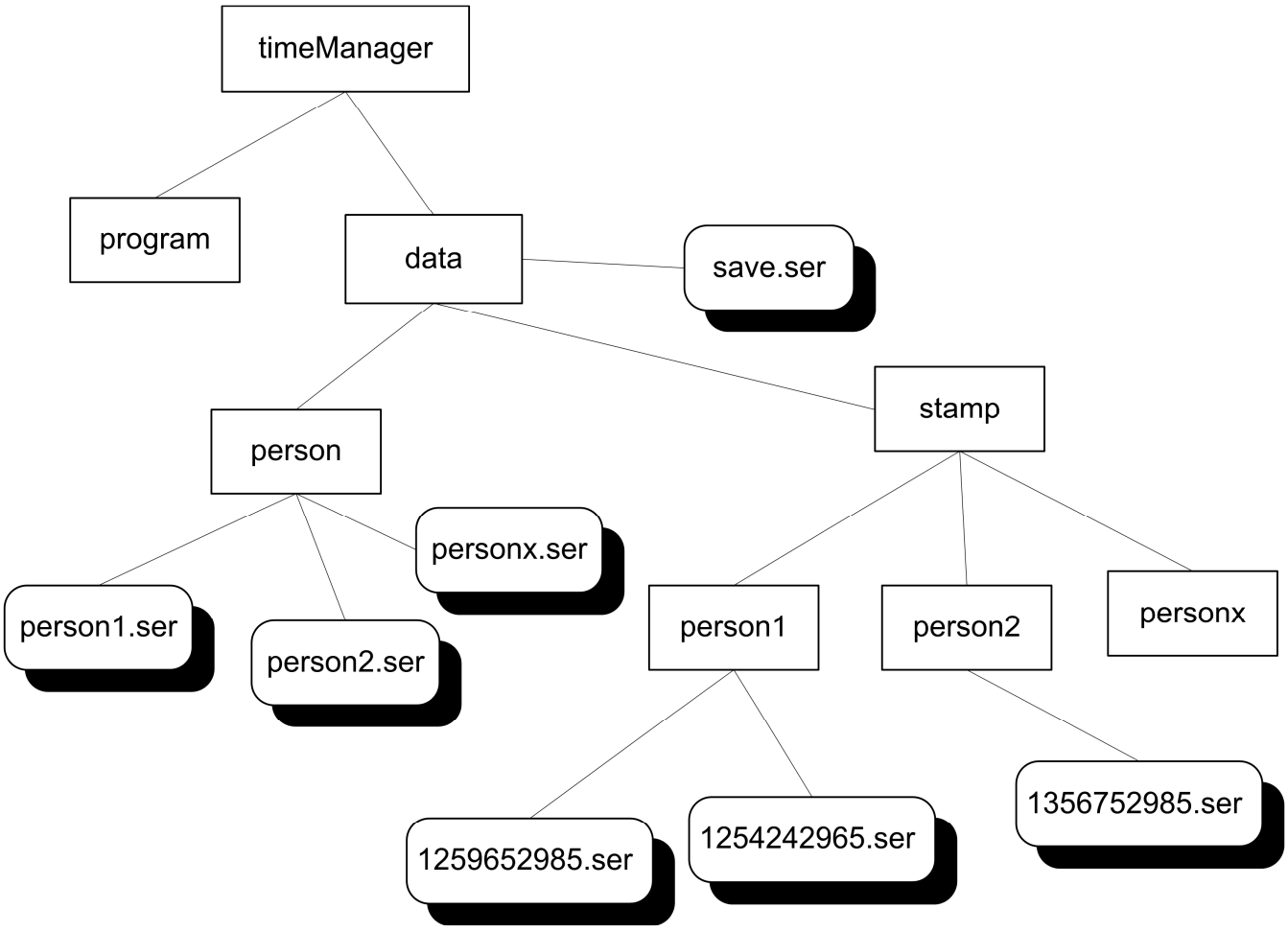
Die Klasse „io“ stellt Funktionen zur Verfügung, die ein Objekt speichern, laden oder Informationen darüber geben. Die Methodenköpfe sollen so aussehen:
public static boolean saveStamp(stamp stp)
public static boolean savePerson(person p)
public static vector<stamp> getStamps(string username)
public static person getPerson(string username)
public static boolean deletePerson(string username)
public static boolean personExists(string username)
public static boolean saveContainer(container cnr)
public static container getContainer()
public static void createDirecotry(String dir)
public static boolean fileExists(String filepath)

4 Programmablauf

<table>
<tr><td colspan="2">Programmstart</td></tr>
<tr><td colspan="2">• Methode Main() wird ausgeführt
• Container-Objekt „save“ wird erstellt (von Festplatte geladen)
• Methode Login() wird aufgerufen
• Im login() wird entweder ein worker oder admin-Objekt aus dem Ordner „data/person“ geladen</td></tr>
<tr><td>Ablauf beim „worker-Objekt“</td><td>Ablauf beim „admin-Objekt“</td></tr>
<tr><td>• Menu des worker-Objekts wird aufgerufen
• → individuelle Wahl</td><td>• Menu des admin-Objekts wird aufgerufen
• → individuelle Wahl</td></tr>
<tr><td>Abmelden</td><td>Abmelden</td></tr>
<tr><td>• Überprüfen, ob ein start-Stamp noch nicht mit einem end-Stemp beglichen wurde.
• Worker-Objekt wird gespeichert.</td><td>• Admin-Objekt wird gespeichert.</td></tr>
<tr><td colspan="2">Programmende</td></tr>
<tr><td colspan="2">• Save-Objekt wird gespeichert</td></tr>
</table>

5 Ordnerstruktur

Die Ordnerstruktur ist so aufgebaut, dass serialisierte Objekte an den geeigneten Orten gespeichert werden können. Dazu gibt es nebst dem Ordner „program“ auch noch einen Ordner „data“. In diesem Ordner gibt es einen Ordner „person“, in dem alle Personen-Objekte gespeichert werden und einen Ordner „stamp“, in dem wiederum für jede Person einen Ordner vorhanden ist, in welchem die Stamp-Objekte gespeichert werden.



5.1 Namenskonvention der Objekte

- Alle Personen-Objekte werden nach ihrem Benutzernamen gespeichert (z.B. person1.ser).
- Alle Stamp-Objekte werden nach dem Timestamp des Stempels gespeichert (z.B. 1356752985.ser).

6 Aufteilung der Arbeit

6.1 Peter Muster

- Klasse „person“
- Klasse „worker“
- Klasse „admin“
- Klasse „container“

6.2 Tobias Muster:

- Klasse „stampmanager“
- Klasse „stamp“
- Klasse „io“
- Klasse „startapplication“