Smart Pointer Projekt – Beschreibung der Klassen

Für den Test im Moped sind genau folgende Dateien zu erzeugen:

license.h

license.cpp

guild.h

guild.cpp

person.h

person.cpp

Zum Test dürfen Sie nur die **Basisimplementierung** mitbringen. Speichern Sie sich also gegebenenfalls einen Zwischenstand bevor Sie die Erweiterung für den Zusatzpunkt implementieren. **Abgabe der Basisimplementierung bis 15.05.22**

1 License

Die Klasse `License` hat folgende **Instanzvariablen**.

string name Name der austellenden Gilde.

unsigned int salary Lohn einer Aufgabe.

unsigned int counter Anzahl der bereits getaetigten Aufgaben.

Die Klasse `License` hat folgende **Konstrukturen und Methoden**.

License(string name, unsigned int salary) Setzt Instanzvariablen. Name darf nicht leer und `salary` muss groesser als 0 sein. Sollte ein Parameter nicht den vorgegebenen Werten entsprechen, ist eine Exception vom Typ `runtime_error` zu werfen.

string get_guildname() const Liefert den Namen der ausstellenden Gilde.

unsigned int get_salary() const Liefert den Lohn einer Aufgabe.

bool valid() const Liefert `true`, falls maximal 3 Aufgaben erst getaetigt wurden, ansonsten `false`.

bool use() Liefert `false`, falls die Lizenz nicht mehr gueltig ist. Ansonsten wird die Anzahl der getaetigten Aufgaben um eins erhoeht und `true` zurueckgeliefert.

ostream& print(ostream& o) const

Die Klasse `License` hat folgendes **Ausgabeformat**.

ostream& print(ostream& o) const Gibt das Objekt auf den ostream `o` aus. Format: [License for `name`, Salary: `salary`, Used: `counter`]

operator<< `License`-Objekte sollen zusätzlich über operator<< ausgegeben werden können. Der operator ist global zu überladen.

Beispiel: [License for City watch, Salary: 5, Used: 0]

2 Guild

Die Klasse `Guild` hat folgende **Instanzvariablen**.

string name Name der Gilde. (Zur Vereinfachung darf davon ausgegangen werden, dass die Namen der Guild-Objekte eindeutig sind.)

unsigned int fee Lizenzgebuehr.

unsigned int salary Gehalt pro Aufgabe.

map<string,shared_ptr<Person>> members Map von Gildenmitgliedern.

Die Klasse `Guild` hat folgende **Konstrukturen und Methoden**.

Guild(string name, unsigned fee, unsigned sal,
const vector<shared_ptr<Person>>& members = {})
Setzt Instanzvariablen, wobei `name` nicht leer sein darf, `fee` und `salary` groesser als 0 sein muessen und keine Person mehr als einmal in `members` vorkommt. Sollte ein Parameter nicht den vorgegebenen Bedingungen entsprechen, ist eine Exception vom Typ `runtime_error` zu werfen.

bool add_member(shared_ptr<Person> p) Fuegt Person `p` der Gilde hinzu, falls nicht schon bereits vorhanden. Liefert `true` bei Erfolg, ansonsten `false`.

bool remove_member(string n) Entfernt Person mit Namen `n` aus der Gilde. Liefert `true` bei Erfolg, ansonsten `false`.

void grant_license(string n) Verkauft Gildenmitglied eine Lizenz der Gilde, sofern sich Person mit Namen `n` eine Lizenz leisten kann (`fee`) und ein Mitglied der Gilde ist. Das Gildenmitglied empfaengt dann die Lizenz. Sollte die Person mit Namen `n` kein Gildenmitglied sein, oder sich die Gebuehr nicht leisten koennen, ist eine Exception vom Typ `runtime_error` zu werfen.

bool offer_job(shared_ptr<Person> p) const Bietet eine Aufgabe Person `p` an, welche diese auch gleich abarbeitet (`work`). Ist `p` Mitglied der Gilde, arbeitet die Person fuer das doppelte Gehalt der Gilde. Eine Lizenz ist fuer Mitglieder nicht notwendig. **Nicht** Gildenmitglieder mit Lizenz, benutzen diese und arbeiten fuer das Gehalt der Lizenz. Alle anderen Personen koennen die Aufgabe nicht bekommen bzw. abarbeiten und die Methode liefert `false`. Ansonsten liefert die Methode `true`.

ostream& print(ostream& o) const

Die Klasse `Guild` hat folgendes **Ausgabeformat**.

ostream& print(ostream& o) const Gibt das Objekt auf den ostream `o` aus.

Format: [`name`, License fee: `fee`, Job salary: `salary`, {$member\_name_0$, $member\_name_1$, ..., $member\_name_n$}]

operator<< `Guild`-Objekte sollen zusätzlich über operator<< ausgegeben werden können. Der operator ist global zu überladen.

Beispiel: [City watch, License fee: 6, Job salary: 5, {Carrot Ironfoundersson, Samuel Vimes}]

3 Person

Die Klasse `Person` hat folgende **Instanzvariablen**.

string name Name der Person. (Zur Vereinfachung darf davon ausgegangen werden, dass die Namen der Person-Objekte eindeutig sind.)

unsigned int wealth

map<string,unique_ptr<License>> licenses Map von aktuellen Lizenzen.

Die Klasse `Person` hat folgende **Konstrukturen und Methoden**.

Person(string name, unsigned int wealth=0) Setzt Instanzvariablen, wobei `name` nicht leer sein darf. Sollte ein Parameter nicht den vorgegebenen Werten entsprechen, ist eine Exception vom Typ `runtime_error` zu werfen.

virtual ~Person() = default;

void work(string guild) Falls eine Lizenz fuer `guild` vorhanden ist und man diese noch benutzen kann, wird die Lizenz benutzt und die Person arbeitet fuer das auf der Lizenz gedruckte Gehalt. Sollte keine Lizenz vorhanden sein, oder nicht mehr benutzbar, ist eine Exception vom Typ `runtime_error` zu werfen.

virtual void work(unsigned int) = 0

void increase_wealth(unsigned int i) Erhoeht `wealth` um `i`.

string get_name() const Liefert `name`.

bool pay_fee(unsigned int i) Sollte das Vermoegen der Person nicht ausreichen, um die Gebuehr `i` zu bezahlen oder `i` 0 sein, ist `false` zu liefern. Ansonsten wird das Vermoegen reduziert um `i` und `true` returniert.

void receive_license(unique_ptr<License> l) Speichert die Lizenz `l` in der Map der Lizenzen der Person. Sollte es bereits eine Lizenz mit dem gleichen Namen geben, ist die alte Lizenz zu zerstoeren.

void transfer_license(string l,shared_ptr<Person> p) Uebergibt die Lizenz mit dem Namen `l` vom `this`-Objekt an Person `p` und entfernt diese aus den Lizenzen vom `this`-Objekt. Sollte die Person (=`this`-Objekt) keine Lizenz mit dem Namen `l` besitzen, ist eine Exception vom Typ `runtime_error` zu werfen.

bool eligible(string l) const Liefert `true` falls eine Lizenz mit Namen `l` vorhanden ist und diese gueltig ist, ansonsten `false`.

virtual ostream& print(ostream& o) const

Die Klasse `Person` hat folgendes **Ausgabeformat**.

ostream& print(ostream& o) const Gibt das Objekt auf den ostream `o` aus.

Format: `name`, `wealth` Coins, {*license_0*, *license_1*, ..., *license_n*}]

operator<< `Person`-,`Superworker`- und `Worker`-Objekte sollen zusätzlich über operator<< ausgegeben werden können. Der operator ist global zu überladen.

Beispiel: Josiah Herbert Boggis, 190 Coins, {[License for City watch, Salary: 5, Used: 0], [License for Thieves guild, Salary: 4, Used: 0]}]

Von der Klasse `Person` werden folgende Klassen abgeleitet.

3.1 Worker

`Worker` ist eine arbeitende Person.

Worker(string, unsigned int=0) Setzt Instanzvariablen durch Konstruktor der Basisklasse.

void work(unsigned int i) Erhoeht `wealth` um `i`.

ostream& print(ostream& o) const Gibt das Objekt auf den ostream `o` aus.

Format: [Worker `Person::print(o)`

3.2 Superworker

`Superworker` ist ein Person, welche eine Gebuehr fuer jede Arbeit verrechnet.

Superworker(unsigned int fee,string,unsigned int=0) Setzt Instanzvariablen durch Konstruktor der Basisklasse. `fee` ist eine zusaetzliche Instanzvariable in der `Superworker`-Klasse.

void work(unsigned int i) Erhoeht `wealth` um `i+fee`.

ostream& print(ostream& o) const Gibt das Objekt auf den ostream `o` aus.

Format: [Superworker `Person::print(o)`

4 Zusatzaufgabe

Erweitern Sie die Klasse `Guild` um ein Logbuch an abgearbeiteten Auftraegen, also ein Objekt, durch welches ersichtlich ist, welche Personen am meisten Geld erwirtschaftet haben, sei es als Gildenmitglied oder als Lizenzarbeiter. Speichern Sie hierzu `weak_ptr` auf die Personen-Objekte und erweitern und aendern Sie bei Bedarf die Klassen.

Beachten Sie, dass Personen-Objekte zerstört werden können. Es muss moeglich sein, sich jederzeit jene noch existierende Person, welche in einer Gilde das meiste Geld erwirtschaftet hat, auszugeben, falls eine existiert.