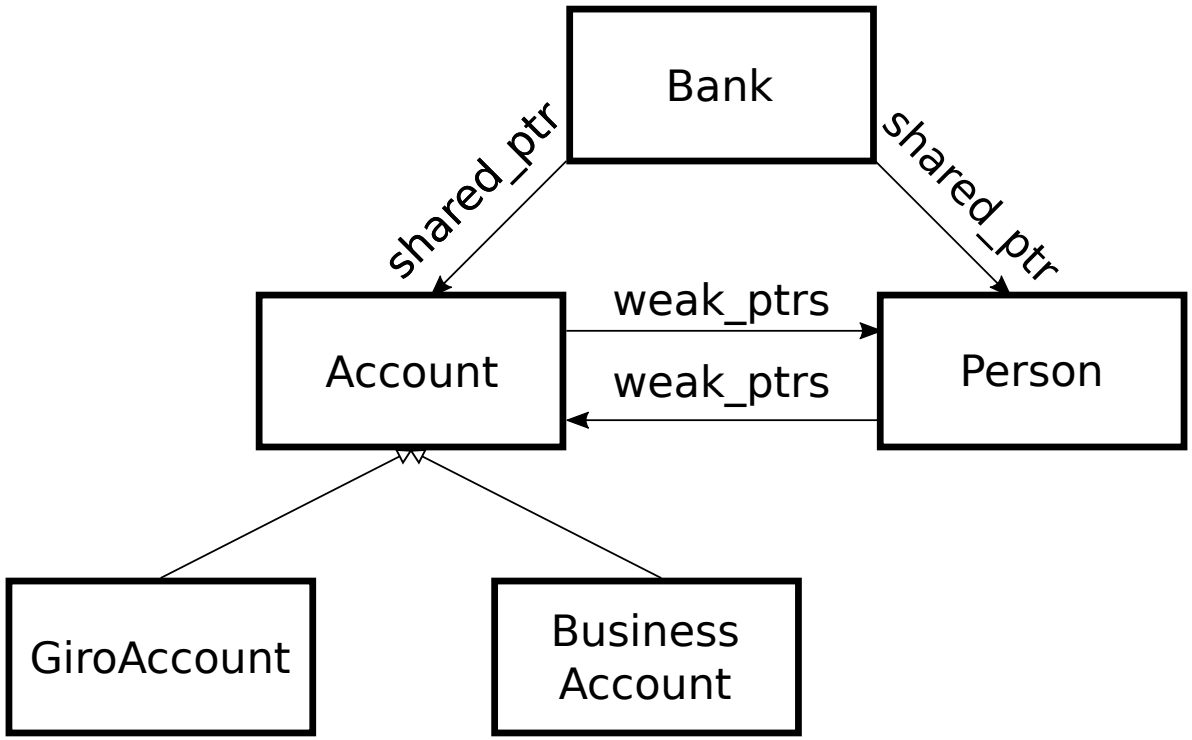
Programmierung 2 VU 051020

Übungseinheit Smart

WS 2020

universität
wien

Bank – Überblick



universität wien

Gaming Platform – Überblick

Zum Testen müssen Sie genau folgende Dateien erzeugen:

player.h (Beinhaltet enum class Mode{Ranked,Unranked};)

player.cpp

game.h

game.cpp

gamekey.cpp

gamekey.h

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 08.12.20

universität wien

Player – Instanzvariablen

string name	Name eines Players.
int mmr	Matchmakingrating eines Players.
shared_ptr<Game> host	Gestartetes Spiel von diesem Spieler.
map<string,weak_ptr<Game>>	Map von Spielen an denen Player teilnimmt.

universität wien

Player – Methoden

Player(string,int) Setzt Instanzvariablen. Name darf nicht leer sein und MMR muss positiv und kleiner als 9999 sein. Sollte ein Parameter nicht gültig sein, ist eine Exception des Typs `runtime_error` zu werfen.

bool host_game(string n,Mode) Wenn `n` leer ist, ist eine Exception des Typs `runtime_error` zu werfen. Sollte Objekt noch kein Game gestartet (**host**) haben, ist ein Game zu erstellen, abhänging von Mode (Ranked/Unranked), das Spiel unter Host einzutragen und `true` zu liefern. Ansonsten `false`.

universität wien

Player – Methoden

bool join_game(shared_ptr<Game> g) Falls Objekt schon in Game-Objekt als Teilnehmer vorhanden, liefert die Methode `false`. Wenn Objekt Game beitreten kann (siehe Teilnahmebedingung in `Game`), wird Objekt bei Game als Teilnehmer eingetragen und in der Map der teilnehmenden Spiele. das Game-Objekt eingetragen, anschließend wird `true` retourniert.

bool leave_game(shared_ptr<Game> g) Liefert `true`, falls Objekt Game verlassen konnte und entfernt Game aus den teilnehmenden Spielen. Retourniert ansonsten `false`.

universität wien

Player – Methoden

string get_name() const Liefert namen des Objekts.

int get_mmr() const Liefert momentanes MMR des Objekts.

shared_ptr<Game> get_host() const Liefert shared_ptr<Game> auf das gestartete Spiel.

void change_mmr(int n) Addiert n zum momemtanen MMR, falls moeglich. Sollte es dabei unter 0 oder ueber 9999 werden, ist eine Exception des Typs `runtime_error` zu werfen.

universität wien

Player – Methoden

vector<weak_ptr<Player>> invite_players(const vector<weak_ptr<Player>>& v)

Versucht Player aus v zum gestarteten Spiel vom this-Objekt einzuladen, also im Game die Player einzuschreiben und bei den Player-Objekten Game in den teilnehmenden Spielen einzutragen. Liefert eine Liste aller weak_ptr welche entweder ungültig waren oder nicht eingeladen werden konnten.

bool close_game() Falls Spiel gestartet ist, soll es freigegeben werden und `true` retourniert werden. Ansonsten `false`.

universität wien

Player – Methoden

ostream& print(ostream& o) const	Gibt das Objekt auf dem ostream aus. Format: [**name**, **mmr**, hosted: (Name von **host**/nothing), games: {**Game**_Name, **Game**_Name, ... }]
operator<<	Player-Objekte sollen zusaetzlich ueber operator<< ausgegeben werden koennen. Der operator ist global zu ueberladen.
Beispiel:	[Heinrich, 20, hosted: nothing, games{Sims 4, Sims 3, Doom}]

universität wien

Game – Instanzvariablen

Um shared pointer vom this-Objekt zu bekommen, erbt Game public von enable_shared_from_this<Game>.
`https://en.cppreference.com/w/cpp/memory/enable_shared_from_this`

string name Name des Spiels.

shared_ptr<Player> host Leiter des Spiels.

map<string,shared_ptr<Player>> Map von teilnehmenden Player-Objekten.

universität wien

Game – Methoden

Game(string,shared_ptr<Player>) Setzt Instanzvariablen. Name darf nicht leer sein und MMR muss positiv und kleiner als 9999 sein. Sollte ein Parameter nicht gültig sein, ist eine Exception des Typs `runtime_error` zu werfen.

string get_name() const Liefert Namen des Objekts.

int get_mmr() const Liefert momentanes MMR des Leiters.

universität wien

Game – Methoden

`void kick_players()` Entfernt alle teilnehmeden Spieler aus dem Objekt.

`bool remove_player(shared_ptr<Player> p)` Entfernt, falls moeglich, p als teilnehmenden Spieler. Liefert *true* falls erfolgreich, ansonsten *false*.

`bool is_allowed(int n) const` Liefert *true* falls n groesser als 90% und kleiner als 110% des MMR des Leiters ist. Ansonsten *false*.

universität wien

Game – Methoden

`bool add_player(const GameKey&,shared_ptr<Player> p)` Falls Player-Objekt mit gleichen Namen schon vorhanden, liefert die Methode *false*. Ansonsten wird p als teilnehmender Spieler hinzugefuegt, falls sein MMR sich um weniger als 10% des Leiters unterscheidet. Im Player-Objekt soll **nicht** das Game-Objekt eingetragen werden. Begruendung und Erklaerung von GameKey siehe naechste Folie.

`bool remove_player(shared_ptr<Player> p)` Entfernt, falls moeglich, p als teilnehmenden Spieler. Liefert *true* falls erfolgreich, ansonsten *false*.

`bool is_allowed(int n) const` Liefert *true* falls n groesser als 90% und kleiner als 110% des MMR des Leiters ist. Ansonsten *false*.

universität wien

Game/Player – Pointer

Die Methode Player::join_game fuegt ein Game-Objekt zu den Spielen hinzu. Damit Game weiss, dass dieses Player-Objekt mitspielt, muss es auch einen Pointer bei seinen teilnehmenden Spielern vermerken. Dies kann die Methode Game::add_game bewaeltigen. Sollte man nun aber im Main einfach Game::add_game aufrufen, kann diese Methode nicht Player::join_game aufrufen, da so sich die beiden Methoden immer wieder aufrufen wuerden ohne Ende. Eine Variante ist die Methode Game::add_game private zu deklarieren, jedoch hat Player auch keinen Zugriff drauf. Die ganze Klasse Player als friend zu deklarieren, gibt der Player-Klasse zu viele Zugriffsrechte auf die Game-Klasse und eine Klassenmethode kann nicht als friend forward deklariert werden.

universität wien

Game/Player – Loesung

Eine Loesung, welche noch immer ein trennen von Interface (Header-File) und Implementierung (CPP-File) ermoeglicht, ist das einfuehren einer neuen Klassen, `GameKey`.

```
#include "player.h"
class Game;
class GameKey{
    GameKey() {} // Private. Implementierung kann auch in GameKey.cpp erfolgen.
    friend bool Player::join_game(std::shared_ptr<Game>);
};
```

Der Konstruktor von GameKey ist **private** deklariert und die Methode Player::join_game als friend deklariert. Somit kann nur die Methode Player::join_game ein GameKey-Objekt erzeugen und innerhalb von Player::join_game Game::add_player(GameKey(), ...) z.B. aufrufen. Es ist mit genug Fantasie noch immer moeglich inkonsistente Objekte zu erzeugen (man koennte noch Kopierkonstruktor, Zuweisungsoperator etc. zu loeschen), aber fuer diesen Kurs ist es so ausreichend.

universität wien

Game – Methoden

shared_ptr<Player> best_player() const Liefert Pointer auf teilnehmendes Player-Objekt mit hoechsten MMR. Bei mehreren Objekten, ist das erste in der Reihenfolge der Map zu liefern.

size_t number_of_players() const Liefert Anzahl der teilnehmenden Spieler.

virtual int change(bool) const = 0 Pure virtual Methode.

shared_ptr<Player> play(size_t i) Itereriert ueber alle teilnehmenden Spieler. Das MMR des i-ten Spielers wird um change(true)+1 geaendert. Das MMR alle anderen wird um 2*change(false) geaendert falls das MMR des Spielers groesser ist, als das MMR des Leiters. Ansonsten um 1*change(false). Retourniert wird ein Pointer auf den i-ten Spieler.

universität wien

Game – Methoden

virtual ostream& print(ostream& o) const — Gibt das Objekt auf dem ostream aus.
Format: [**name**, **host->Name**, **host->MMR**, player: {[**PlayerName**, **PlayerMMR**], [**PlayerName**, **PlayerMMR**], ... }]

operator<< — Game-Objekte sollen zusaetzlich ueber operator<< ausgegeben werden koennen. Der operator ist global zu ueberladen.

Beispiel: [DotA 2, Juliane, 558, player: [Heinrich, 575], [Helmut, 582], [Juliane, 558]]

universität wien

Game – Abgeleitete Klassen

RGame

RGame(string,shared_ptr<Player>) Setzt Instanzvariablen durch Konstruktor der Basisklasse.

int change(bool x) const Liefert 5 falls `true`, ansonsten -5.

virtual ostream& print(ostream& o) const Gibt das Objekt auf dem ostream aus.
Format: Ranked Game: **Game->Print**

UGame

UGame(string,shared_ptr<Player>) Setzt Instanzvariablen durch Konstruktor der Basisklasse.

int change(bool) const Liefert 0.

virtual ostream& print(ostream& o) const Gibt das Objekt auf dem ostream aus.
Format: Unranked Game: **Game->Print**

universität wien

Testen, Testen, Testen

Schreiben Sie ein eigenes main.cpp in welchem Sie

- Objekte jeden Typs anlegen und prüfen ob beispielsweise Fehlermeldungen korrekt ausgelöst werden.
- Methoden aufrufen und deren Ergebnisse (händisch) prüfen.

Um weitere Testmöglichkeiten zu erhalten, können Sie

- Ihre main.cpp Files mit Studienkolleg*innen austauschen.
- Moped: heb9b

Gaming Platform: Zusatzpunkt – Hinweise

- Sofern Sie den Zusatzpunkt erhalten möchten, beachten Sie bitte, dass in diesem Fall sowohl für die Basisimplementierung, als auch für die Zusatzimplementierung eine automatische **Plagiatsüberprüfung** durchgeführt wird.
- Projektpunkte tragen **nicht** zu den, für einen positiven Abschluss erforderlichen Testpunkten bei, führen aber im Fall einer positiven Bewertung gegebenenfalls zu einer Verbesserung der Note.
- **Abgabe des Zusatzes bis 13.01.21**
- Denken Sie daran sich Ihre Basisimplementierung für den Test zwischen zu speichern bevor Sie mit dem Zusatz beginnen.

universität wien

Gaming Platform: Zusatzpunkt – Anforderung

Für den Zusatzpunkt auf das Smart Pointer Projekt, ist Folgendes zu implementieren.

universität wien