Prof.Dr.H.P.Weber	Praktische Prüfung PAD1 WS 14/15	h_da HOCHSCHULE DARMSTADT UNIVERSITY OF APPLIED SCIENCES fbi FACHBEREICH INFORMATIK

Name: MatNr: User:

Hinweis: Denken Sie während der Klausur auch an die **Datensicherung!**
Empfehlung: Arbeiten Sie in kleinen Schritten und sorgen Sie dafür, dass Sie jederzeit ein lauffähiges Anwendungssystem haben.

Allgemeine Vorgaben:

- **Benutzen Sie ausschließlich MS Visual Studio Professional 2013.**
- Verwenden Sie möglichst keine globalen Variablen oder Objekte und nur solche, die als **const** deklariert sind.
- Programmieren Sie C++-standardkonform (z.B. keine Verwendung von **<conio.h>**, kein Aufruf von **system**, keine Ein-/Ausgabe im C-Stil!)
- Verwenden Sie die vorgegebenen Bezeichner; zusätzliche eigene Bezeichner sind 'sprechende Bezeichner'.
- Beachten Sie das 'Principle of least privilege' (z.B. 'const-correctness').
- Die öffentliche Schnittstelle der Klassen darf durch **operator<** und **operator==**-Funktionen erweitert werden.
- Alle Variablen/Objekte werden zur Übersetzungszeit erzeugt und direkt mit ihrem Namen angesprochen, d.h. in Ihrem Programm kommt kein **new**-Operator und keine Zeiger-Variable vor.
- Trennung von Klassendefinitionen und –implementierungen in .h- und .cpp-Dateien.

Zur Bewertung:
Insgesamt können Sie 100 Punkte bekommen.
Die Note 'sehr gut' (1,0 und 1,3) wird für mindestens 93 Punkte vergeben.
Bestanden haben Sie mit mindestens 50 Punkten.
Werden die 'Allgemeinen Vorgaben' nicht eingehalten, gibt es Punktabzug.

Aufgabenstellung:
Schreiben Sie ein Programm, das durch Simulationen mit unterschiedlichen Randbedingungen den Gewinn bestimmt, der bei verschiedenen Spielstrategien im Roulette erzielt wird. Verwenden Sie dazu die Klasse **Roulette** und die unterstützenden Klassen **Player** und **Number** (s. Klassendiagramme):

Roulette
+ nrP: int **+ nrN: int** **- players: array<Player,nrP>** **- roulette: array<Number,nrN>**
+ Roulette(int) **+ makeBets(): void** **+ play(): void** **+ sort(): void** **+ showPlayers(): string** **+ showNumbers(): string**

Player
- id: int **- bet: char** **- number: Number** **- budget: int** **- moneyBet: int** **- playing: bool**
+ Player(int, int) **+ takeFromBudget(int): bool** **+ addToBuget(int): void** **+** <*erforderliche setter/getter*>

Number
- value: int
+ Number(int) **+ isEven(): bool** **+ isLow(): bool** **+ toString(): string** **+** <*erforderliche setter/getter*>

Ziel beim Roulette ist es, in jedem einzelnen Spiel vorherzusagen, auf welche der Zahlen 1 bis 36 die Kugel fallen wird oder wenigstens eine Eigenschaft der Gewinnzahl wie gerade/ungerade oder niedrig/hoch vorherzusagen. Bei richtiger Vorhersage erhält der Spieler einen Gewinn von der Bank ausgezahlt (dieser schließt seinen vor dem Spiel eingesetzten Geldbetrag mit ein). Fällt die Kugel auf die Zahl 0, gewinnt keiner der Spieler, sondern die Bank.

Für die Datenelemente und Elementfunktionen gilt:

- Der Konstruktor von **Number** setzt das Datenelement **value** auf eine von außen übernommene Zahl von 0 bis 36. Die Elementfunktion **isEven** gibt **true** bei geraden Zahlen zurück und **false** bei ungeraden, **isLow** gibt **true** bei Zahlen kleiner als 19 zurück und **false** bei Zahlen größer als 18.
- **toString** soll den Wert von **value** als Zahl sowie die Information von **isEven** als ‚gerade' bzw. ‚ungerade' und von **isLow** als ‚tief' bzw. ‚hoch' zusammen in einer Zeile zurückgeben (im Fall von 0 nur den Wert von **value**).
- Jeder **Player** hat eine eindeutige Identifikationsnummer **id** und einen Buchstaben **bet**, der festhält, wie der Spieler im aktuellen Spiel gesetzt hat (**g** für ‚gerade', **u** für ‚ungerade', **t** für ‚tief', **h** für ‚hoch' und **z** für eine bestimmte Zahl). Falls der Spieler auf eine bestimmte Zahl gesetzt hat, wird diese in **number** festgehalten. **budget** enthält den Geldbetrag, den der Spieler momentan besitzt, **moneyBet** ist sein Einsatz im aktuellen Spiel und **playing** gibt an, ob der Spieler (noch) am Spiel teilnimmt

- Der Konstruktor von **Player** setzt zumindest **id** und **budget** auf die von außen als Parameter übergebenen Werte und **playing** auf **true**.
- **takeFromBudget** soll den als Parameter übergebenen Wert vom **budget** abziehen, falls möglich. In diesem Fall wird **true** zurückgegeben. Falls das **budget** zu niedrig ist, wird nichts abgezogen und **false** zurückgegeben.
- **addToBudget** soll den als Parameter übergebenen Wert zum **budget** addieren.
- Es nehmen 4 Spieler am Roulette teil, d.h. die (konstante!) Klassenvariable **nrP** hat den Wert **4**. Ein Roulette enthält die Zahlen 0 bis 36, d.h. **nrN** hat den Wert **37**.
- Der Konstruktor von **Roulette** übernimmt von außen einen Wert, mit dem das **budget** aller 4 **Player** einheitlich initialisiert wird. Außerdem vergibt er eindeutige Identifikationsnummern an die 4 **Player** und erzeugt die 37 **Number**-Objekte für das **roulette**-Array.
- **makeBets** legt fest, wie die 4 Spieler setzen und welchen Geldbetrag sie einsetzen. Im Rahmen der Simulation soll dies in jeder Spielrunde gleich ablaufen: Einer der Spieler (aber immer der gleiche!) setzt immer ‚gerade', ein anderer immer ‚tief', der nächste immer die Zahl 23 und der letzte immer eine zufällig gewählte Zahl von 1 bis 36. Alle vier setzen immer einen Betrag von 100 Euro, um diesen Betrag vermindert sich also ihr **budget**. Ein Spieler setzt nur, solange er noch mitspielt (s. unten).
- **play** realisiert eine Spielrunde: Eine zufällige Zahl aus dem Bereich 0 bis 36 wird bestimmt und für die 4 Spieler wird ermittelt, ob sie gewonnen haben und der Gewinn wird ausgezahlt. Der Gewinn besteht beim Setzen auf ‚gerade' oder ‚tief' aus dem doppelten Einsatz und beim Setzen auf eine Zahl aus dem 36-fachen Einsatz. Falls das **budget** eines Spielers auf 0 sinkt und er dann verliert, kann er nicht mehr mitspielen (d.h. sein **playing** wird auf **false** gesetzt).
- **sort** sortiert die 4 Spieler innerhalb des Arrays **players** absteigend nach ihrem **budget**. Sie können ein Sortierverfahren ihrer Wahl verwenden. Es ist erlaubt (und sinnvoll) **Player** hierfür durch einen Kleiner-Operator zu ergänzen.
- **showPlayers** gibt für jeden der 4 Spieler Identifikationsnummer und aktuelles **budget** als **string** zurück.
- **showNumbers** gibt alle 37 Zahlen des **roulette** mit allen ihren Eigenschaften als **string** zurück (s. **toString** von **Number**).

Das **Anwendungsprogramm (main)** soll die Simulationen durchführen. Dazu sollen der Reihe nach folgende Aktionen ablaufen (es soll also kein Menü realisiert werden):

- Erzeugung eines **Roulette**-Objektes (**budget** der Spieler: 1000 Euro) und Ausgabe aller 37 Zahlen mit ihren Eigenschaften.
- Durchführung von 20 Spielrunden mit diesem **Roulette**-Objekt. Dabei sollen nach jeder Runde die Identifikationsnummern und die **budget**s der 4 Spieler ausgegeben werden.
- Erzeugung eines neuen **Roulette**-Objektes (**budget** der Spieler: 1 000 Euro) und Durchführung von 100 Spielrunden. Erst nach den 100 Runden sollen die **id**-nummern und die **budget**s der 4 Spieler ausgegeben werden.
- Erzeugung eines neuen **Roulette**-Objektes (**budget** der Spieler: 10 000 Euro) und Durchführung von 100 Spielrunden. Erst nach den 100 Runden sollen die **id**-nummern und die **budget**s der 4 Spieler ausgegeben werden.
- Erzeugung eines neuen **Roulette**-Objektes (**budget** der Spieler: 10 000 Euro) und Durchführung von 1 000 Spielrunden. Erst nach den 1 000 Runden sollen die Identifikationsnummern und die **budget**s der 4 Spieler ausgegeben werden und zwar diesmal absteigend nach **budget**s sortiert.

Ergebnis:

Als Ergebnis erwarte ich in jedem Fall ein **ausführbares** Anwendungssystem.
Speichern Sie alle Ihre Projektdateien mit Ausnahme des Debug- und ipch-Verzeichnisses in Ihrem persönlichen Netzverzeichnis. Das Anwendungssystem muss sich mit diesen Dateien auf "Knopfdruck" erzeugen lassen.
Löschen Sie Ihr Projekt am Klausurende **nicht** und fahren Sie den Rechner **nicht** herunter.

Lassen Sie das Aufgabenblatt (mit Namen, Matrikel- und Usernummer) an Ihrem Platz liegen.

Bewertung (Punkte):	**maximal**	**erreicht**
Klassendefinitionen	14	
Implementierung **Number**	12	
Implementierung **Player**	12	
Konstruktor **Roulette**	6	
makeBets	10	
play	15	
sort	9	
showPlayers	5	
showNumbers	5	
main	12	

Gesamtpunkte:	**Note:**