# Objektorientierte Sprachen

**Sommersemester 2020 - Erste Prüfung**

## An- und Abgabe

Schreiben Sie ein textbasiertes Ping-Pong-Spiel, in dem zwei Spieler-Objekte neun Runden lang gegeneinander spielen. Eine Runde endet, sobald ein Objekt einen Punkt erzielt. Nach den neun Runden gewinnt das Objekt, das mehr Punkte erzielt hat.
Sie haben dazu eine Grundstruktur vorgegeben, die bereits die komplette main-Funktion enthält, sowie die Header- und CPP-Dateien für das IPlayer-Interface bereitstellt. Erweitern Sie nun diese Grundstruktur, indem Sie das Interface ausbauen und zwei Klassen implementieren, die vom Interface erben.

**Es werden nur Abgaben benotet, die auch kompilierfähig sind. Kommentieren Sie zur Not die Funktionen in der main-Funktion aus, die Sie nicht implementiert haben.**

**Laden Sie Ihre Abgabe in einem ZIP-Archiv verpackt auf Moodle hoch, fehlerhaft oder unvollständig hochgeladene Abgaben können nicht nachgereicht werden.**
**Die Abgabe ist bis XX:XX möglich.**

Die Grundstruktur ist im Code durch Kommentare erklärt.

# Aufgaben

## Interface IPlayer

Definieren Sie folgende **rein virtuelle** Funktionen für das Interface:

**attack:** Die Funktion soll einen booleschen Wert zurückgeben und einen Pointer auf ein anderes IPlayer-Objekt als Parameter verwenden.

**beAttacked:** Die Funktion soll einen double-Wert zurückgeben und hat keine Parameter.

Implementieren Sie weiters den **Operator** $>$ für das Interface. Er soll einen booleschen Wert zurückgeben und einen Pointer auf ein anderes IPlayer-Objekt als Parameter verwenden. Ist der points-Wert des Objekts größer als der points-Wert des zweiten Operanden, wird true zurückgegeben, ansonsten false.
Wird ein NULL-Pointer als Parameter übergeben, soll eine invalid_argument-Exception geworfen werden.

### Benotungsaspekte

| Aspekt | Bewertung |
|---|---|
| Rein virtuelle Methoden | 10% |
| $>$ Operator | 10% |
| Exception | 4% |
| **Gesamt** | 24% |

## Klasse PlayerA

Definieren Sie die Klasse PlayerA, die vom Interface IPlayer erben soll, aufgeteilt in separate Header- und CPP-Dateien und implementieren Sie folgende Funktionen für die Klasse:

**Konstruktor:** Der Konstruktor soll als Parameter einen String mit dem Namen des Objekts verwenden und die Attribute der Klasse folgendermaßen belegen:

| Attribut | Wert |
|---|---|
| attackBonus | 0 |
| attackPoints | 40 |
| defensePoints | 60 |
| name | Parameter |
| points | 0 |
| stamina | 100 |

**attack:** Zuerst erhöht sich der attackBonus-Wert des Objekts um einen zufälligen Wert zwischen 0 und 4. Danach wird der Angriffswert (tmpAttackValue) dieses Angriffs folgendermaßen berechnet:

$$\text{tmpAttackValue} = (\text{attackBonus} + \text{attackPoints}) * \text{opponent}\rightarrow\text{beAttacked}()$$

Wenn der Angriffswert dann höher als der defensePoints-Wert des opponent-Objekts ist erzielt das angreifende Objekt einen Punkt. Die Punkte des angreifenden Objekts werden um 1 erhöht, und dann wird true zurückgegeben, ansonsten false.
Wird ein NULL-Pointer als Parameter übergeben, soll eine invalid_argument-Exception geworfen werden.

**beAttacked:** Zuerst wird der stamina-Wert des Objekts um 3 gesenkt, der Wert kann dabei nicht unter 0 sinken.
Je nachdem durch welche Zahl der stamina-Wert dann teilbar ist, wird ein anderer Wert zurückgegeben. Dabei gilt, dass wenn der stamina-Wert durch eine Zahl (zum Beispiel 7) teilbar ist, andere Teiler (5, 2) nicht mehr geprüft werden, dabei wird 7 vor 5 vor 2 geprüft.

| stamina-Wert teilbar durch | Rückgabewert |
|---|---|
| 7 | 5 |
| 5 | 3 |
| 2 | 1 |
| ansonsten | 0 |

**Benotungsaspekte**

| Aspekt | Bewertung |
|---|---|
| Header-/CPP-Dateien | 4% |
| Vererbung | 5% |
| Konstruktor | 5% |
| attack-Funktion | 10% |
| Exception | 4% |
| beAttacked-Funktion | 10% |
| **Gesamt** | 38% |

### Klasse PlayerB

Definieren Sie die Klasse PlayerB, die vom Interface IPlayer erben soll, aufgeteilt in separate Header- und CPP-Dateien. **Diese Klasse hat ein zusätzliches Attribut, einen booleschen Wert "confused"**. implementieren Sie folgende Funktionen für die Klasse:

**Konstruktor:** Der Konstruktor soll als Parameter einen String mit dem Namen des Objekts verwenden und die Attribute der Klasse folgendermaßen belegen:

| Attribut | Wert |
|---|---|
| attackBonus | 0 |
| attackPoints | 45 |
| defensePoints | 55 |
| confused | false |
| name | Parameter |
| points | 0 |
| stamina | 100 |

**attack:** Zuerst besteht eine Chance von 30% das sich confused ins Gegenteil verkehrt, sprich wenn confused true ist, wird es false und umgekehrt.
Ist das Objekt dann nicht confused erhöht sich der attackBonus-Wert des Objekts um den fünffachen Rückgabewert der opponent→beAttacked-Funktion.
Danach wird der Angriffswert (tmpAttackValue) dieses Angriffs folgendermaßen berechnet:

$$\text{tmpAttackValue} = \text{attackBonus} + \text{attackPoints}$$

Wenn der Angriffswert dann höher als der defensePoints-Wert des opponent-Objekts ist erzielt das angreifende Objekt einen Punkt. Die Punkte des angreifenden Objekts werden um eins erhöht, und dann wird true zurückgegeben, ansonsten false.
Wird ein NULL-Pointer als Parameter übergeben, soll eine invalid_argument-Exception geworfen werden.

**beAttacked:** Ist das Objekt verwirrt, also confused gleich true, wird der stamina-Wert um ein Fünftel seines derzeitigen Werts gesenkt, ansonsten wird er um ein Zwanzigstel seines derzeitigen Werts gesenkt.
Wenn der stamina-Wert dann größer als 30 ist, wird 0 zurückgegeben, ansonsten 1.

**Benotungsaspekte**

| Aspekt | Bewertung |
|---|---|
| Header-/CPP-Dateien | 4% |
| Vererbung | 5% |
| Konstruktor | 5% |
| attack-Funktion | 10% |
| Exception | 4% |
| beAttacked-Funktion | 10% |
| **Gesamt** | 38% |

# Benotung

| | |
|---|---|
| $\geq 88\%$ | Sehr gut |
| $\geq 75\%$ und $< 88\%$ | Gut |
| $\geq 63\%$ und $< 75\%$ | Befriedigend |
| $\geq 50\%$ und $< 63\%$ | Genügend |
| $< 50\%$ | Nicht genügend |