ulm university universität
uulm

Institut für Numerische Mathematik

Dr. Andreas F. Borchert

24. Mai 2018
Blatt 3

Objektorientierte Programmierung mit C++ (SS 2018)

Abgabe bis zum 7. Juni 2018, 16:00 Uhr

Lernziele:

- Entwicklung einer abstrakten Klasse und zugehöriger polymorpher Implementierungen
- Dynamisches Nachladen polymorpher Implementierungen

Aufgabe 4: Polymorphe Spieler

Im Rahmen der Aufgabe 1 (Blatt 1) wurde das Nim-Spiel mit n Haufen implementiert. Der durch das Hauptprogramm implementierte Spielablauf in *Nim.cpp* sah etwa so aus:

```
while (!game.finished()) {
   // print current state ...
   // determine next move
   NimMove move;
   if (game.get_next_player() == NimGame::PLAYER1) {
      // human player
      // ask human player for a move...
   } else {
      // computer
      // look for a move that yields a nim value of 0, if possible
   }
   game.execute_move(move);
}
```

Dies ist natürlich unbefriedigend, da die beiden Spieler-Implementierungen so ein fester Bestandteil des Hauptprogramms bilden. Es erscheint daher erstrebenswert,

- eine abstrakte Klasse *NimPlayer* zu entwickeln,

- mehrere Erweiterungen dieser Klasse zu implementieren wie beispielsweise *HumanNimPlayer*, *RandomNimPlayer* oder *OptimalNimPlayer*,
- und diese beim Start des Programms auswählbar zu machen, z.B. indem die gewünschten Implementierungen dynamisch nachgeladen werden.

Im Rahmen dieser Aufgabe ist dies umzusetzen, wobei Sie mindestens zwei Implementierungen entwickeln sollten, wozu auch *HumanNimPlayer* gehören muss.

Hinweise: Denken Sie darüber nach, wie das Ende der Eingabe bei *HumanPlayer* zu behandeln ist. Hier ist es keine Lösung, die Ausführung zu beenden. Stattdessen ist es geschickter, einen Zug einzuführen, bei dem ein Spieler „aufgibt". Bei einer Aufgabe hat ein Spieler sofort verloren.

Analog wie beim *DynFunctions*-Beispiel aus der Vorlesung werden Sie *construct*-Funktionen bei den einzelnen Implementierungen der Spieler benötigen. Singleton-Objekte reichen für den Zweck dieser Übungsaufgabe vollkommen aus.

Eine *Registry* wie im *DynFunctions*-Beispiel benötigen Sie hier nicht, da Sie nur zwei Spieleteilnehmer laden. Sie können diese Ladefunktionalität z.B. in *Nim.cpp* direkt integrieren.

Wenn Sie in den dynamisch nachzuladenden Implementierungen Methoden verwenden, die bereits im zu startenden Programm eingebunden sind (wie etwa die Methoden von *NimMove* und *NimGame*, dann muss das Hauptprogramm mit der Option „-rdynamic" zusammengebaut werden. Ohne diese Option gibt es Fehler bei der Ausführung dynamisch nachgeladener Objekte. Die Meldung lautet dann etwa „relocation error", gefolgt von „referenced symbol not found". Um die korrekte Übersetzung zu erleichtern, können Sie gerne dieses *Makefile* verwenden und anpassen.

Wenn Sie Ihre Lösung einreichen, sollten Sie diese bitte zunächst mit Hilfe von *tar* verpacken und komprimieren:

```
theon$ tar cvfz Nim.tar.gz *.?pp [mM]akefile
```

Sie können dann Ihre Lösung wieder mit *submit* einreichen:

```
theon$ submit cpp 4 Nim.tar.gz
```

Viel Erfolg!