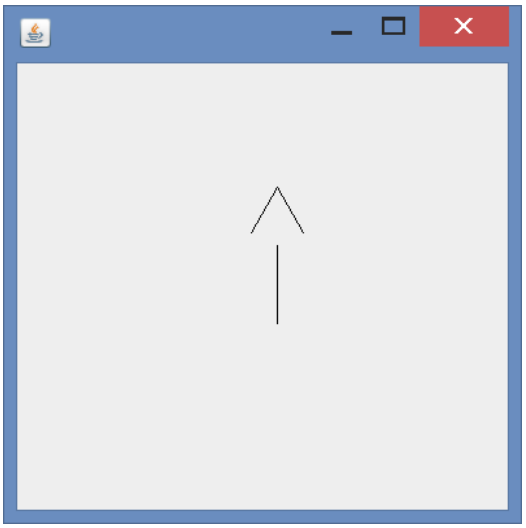
Arbeitsblatt Undo / Redo und Makros im LogoInterpreter

Auf dem File Server finden Sie im Wochenprojekt *11_Command.zip* einen Interpreter für *Turtle Graphics* mit der Sprache *Logo*, einer einfachen Sprache, die auch verwendet wird, um Kindern das Programmieren näher zu bringen. Wenn Sie das Programm starten, können Sie anschliessend in der Konsole Befehle eintippen mit der Sie eine *Turtle* mit einem Stift über die Zeichenfläche bewegen.

Beispielsweise führt der folgende Dialog dann zum daneben gezeigten Bild:

```
Starting interpreter...
forward 60
Moving 60 steps.
penup
Lifting pen up.
forward 10
Moving 10 steps.
left 90
Rotating 90 degrees left.
forward 20
Moving 20 steps.
pendown
Putting pen down.
right 120
Rotating -120 degrees left.
forward 40
Moving 40 steps.
right 120
Rotating -120 degrees left.
forward 40
Moving 40 steps.
```



Die Klasse *LogoInterpreter* im Package *logo* enthält in der Methode *run* eine Schleife, welche die Benutzereingaben verarbeitet. Dabei wird jeweils ein Command-Objekt von einem Parser erzeugt:

```
Command command = parser.parse(scanner);
```

Diese so erzeugten *Commands* werden am Ende der Schleife ausgeführt:

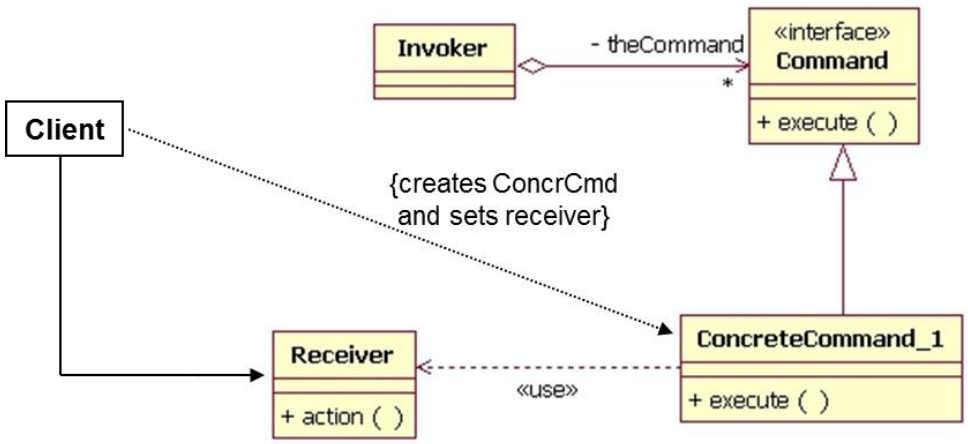
```
command.execute();
```

Der Interpreter enthält auch eine weitere Anwendung für solche Command-Objekte: Das *RepeatCommand*. Ein solches Kommando erzeugen Sie mit einem Befehl wie:

```
repeat 5 forward 10
Repeating 5 times Moving 10 steps.
```

Aufgaben

a) Versuchen Sie herauszufinden, wie das *repeat*-Kommando funktioniert, indem Sie die Klasse *RepeatCommand* im Package *logo.commands.turtle* studieren. Wie wird das Command Pattern in dieser Klasse angewendet? Identifizieren Sie für diese Situation *Invoker* und *Receiver* usw. für das wiederholt auszuführende Kommando und notieren Sie Namen konkreter Klassen bzw. Interfaces in das allgemeine Command Pattern-Klassendiagramm:



b) Der Logo-Interpreter ist für eine Erweiterung mit *undo*- und *redo*-Befehlen vorbereitet. Diese funktionieren aber noch nicht, weil sie einen *HistoryManager* benötigen, der im Moment noch leer in der Klasse *StdHistoryManager* im Package *logo* implementiert ist. Diese Klasse enthält eine Reihe *TODO*-Markierungen und Erklärungen der jeweils zu implementierenden Methoden.

Die Implementierung der Turtle-Graphik ist speziell, denn es wird kein eigentliches Modell dieser Graphik gespeichert, in dem man Linien hinzufügen oder wieder entfernen könnte. Statt Linien zu löschen wird gerade die vom *HistoryManager* verwaltete Sequenz von *Commands* benutzt, um die Graphik von Grund auf neu zu erzeugen. Dies tut die Methode *repaint* im Logo-Interpreter.

Testen Sie die Befehle *undo* und *redo* nach der Implementierung der fehlenden Methoden in der Klasse *StdHistoryManager* z.B. mit den folgenden beiden Sequenzen. Passiert, was Sie erwarten?

```
repeat 3 forward 20
undo
```

```
forward 50
forward 100
undo
left 90
redo     // nothing should happen
undo
undo
```

c) Im Logo-Interpreter kann man Makros aufzeichnen und anschliessend immer wieder ausführen. Damit wird das System für Benutzer programmierbar.

Der Interpreter kennt dazu bereits folgende Befehle:

`macrorecord` *`name`* beginnt ein Makro mit dem Namen *name* aufzuzeichnen

`macrosave` beendet die Aufzeichnung und speichert das Makro unter seinem Namen

`macrorun` *`name`* führt das zuletzt mit dem Namen *name* aufgezeichnete Makro aus

Diese Befehle funktionieren im Moment noch nicht. Es muss zuerst noch ein *MacroManager* implementiert werden. Ein solcher ist bereits in der Klasse *StdMacroManager* im Package *logo* vorbereitet. Diese Klasse enthält eine Reihe *TODO*-Markierungen und Erklärungen der jeweils zu implementierenden Methoden.

Zum Speichern eines Makros können Sie eine Instanz der Klasse *CompositeCommand* verwenden, die Sie im Package *logo.commands.turtle* finden.

Testen Sie den *MacroManager* mit folgenden Befehlen, die einen Stern zeichnen, wenn alles richtig funktioniert. Dieses Script ist auch im Projekt unter dem Namen *script2.txt* abgelegt.

```
macrorecord dash penup forward 10 pendown forward 10
macrosave
macrorecord dashline forward 10 repeat 2 macrorun dash
macrosave
macrorecord starlineAndTurn macrorun dashline left 60 macrorun dashline
right 120 macrorun dashline left 60 macrorun dashline right 120
macrosave
repeat 3 macrorun starlineAndTurn
```

Erweitern Sie nun noch den *MacroManager* so, dass während der Aufzeichnung von Makros eine Vorschau des Makros in roter Farbe angezeigt wird. Dazu müssen jeweils die zur Aufzeichnung übergebenen *Commands* auch direkt einmal ausgeführt werden. Zum Einstellen der Farbe können Sie die Methode *interpreter.setColor* benutzen. Am Ende der Aufzeichnung muss diese Vorschau wieder verschwinden. Dazu können Sie – wie bei *undo* in Aufgabe b) – die *repaint*-Methode des Interpreters benützen, denn der Interpreter zeichnet die *Commands*, die für Makros verwendet werden, nicht in der History auf.