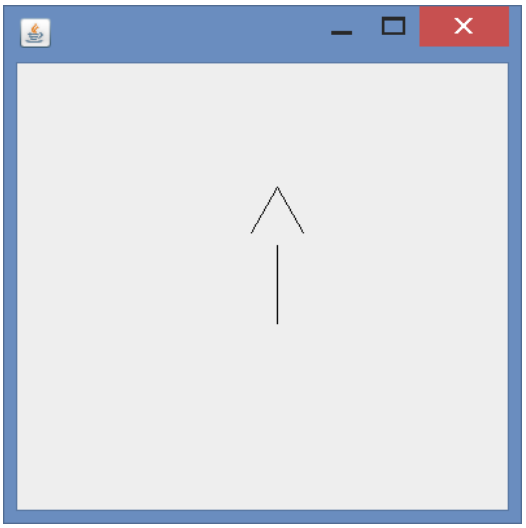
Worksheet Undo / Redo and Macros for Logo

In this week's project *11_Command.zip* on the file server (active directory) you find an interpreter for the programming language *Logo* which allows conducting a turtle. The programming language *Logo* is used to introduce Kids to the world of programming. If you start the program, you can enter commands in the console which conduct a *Turtle* with a pen over the drawing area.

For example, the following dialog results in the image shown aside:

```
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.
```



Class *LogoIntepreter* in package *logo* contains in method *run* a loop that processes user input. The commands entered by the user are turned into command objects by a parser:

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

These commands are then executed inside the loop with

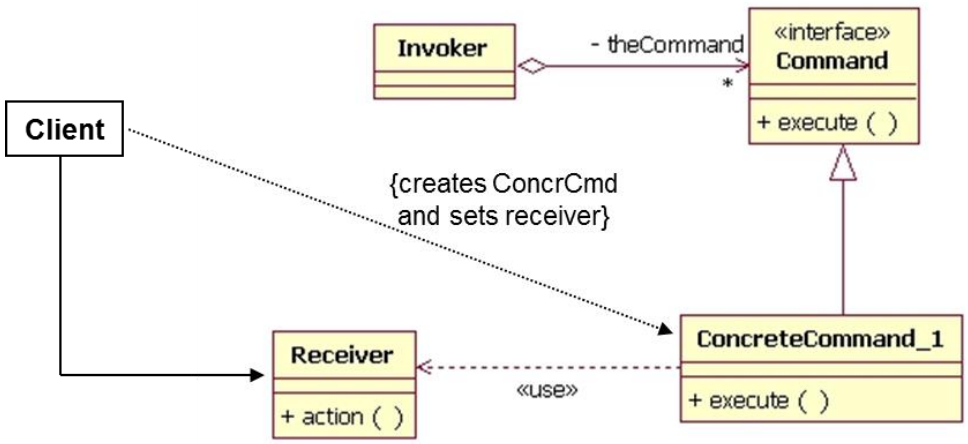
```
command.execute();
```

The interpreter contains an additional application of the command pattern: the repeat command. This command allows to execute a command several times. An instance of a *RepeatCommand* can be created with a command like the following:

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

Tasks:

a) Scrutinize class *RepeatCommand* defined in the package *logo.commands.turtle* and try to find out how it works. How is the *Command Pattern* applied? Identify for this application invoker and receiver for the repeatedly executed command and add the names of concrete classes and interfaces into the general Command-Pattern class diagram:



b) The *Logo* interpreter is prepared to support *undo* and *redo* commands. However, these commands do not yet work because they need a *HistoryManager*, which is currently implemented with empty methods in class *StdHistoryManager* in package *logo*. This class contains a bunch of TODO markers and explanations of what needs to be done.

The implementation of the turtle graphic is special, because no actual model is used to store the graphics by adding and removing lines. Instead of deleting lines, the sequence of commands managed by the *HistoryManager* is used to create the graphics from scratch. This is done by method *repaint* defined in class *LogoInterpreter*.

Test your implementation of class *StdHistoryManager* with the following two sequences. Is happening what you are expecting?

```
repeat 3 forward 20
undo
```

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

c) The *Logo* interpreter can be used to record macros which then can be executed over and over again. This makes the system programmable for users.

The interpreter already supports the following commands:

`macrorecord` *name*	starts recording a new macro with the given name
`macrosave`	stops the recording and stores the macro under its name
`macrorun` *name*	executes the macro recorded at last under the given name

These commands do currently not yet work. A *MacroManager* must be implemented first. An implementation of such a manager is prepared in class *StdMacroManager* in package *logo*. This class contains a bunch of *TODO* markers and explanations of what needs to be done.

To represent a macro, you can use an instance of class *CompositeCommand* whith you find in package *logo.commands.turtle*.

Test your implementation with the following commands that draw a star (if everything works properly). This script is also stored in this week's project and additionally on the file server under the name *script2.txt*.

```
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
```

Now extend your implementation of the *MacroManager* so that a preview of the macro is displayed during the recording in red color. To do so, the commands passed for recording to method *addCommand* must be executed as well. To set the color, use method *interpreter.setColor*. At the end of the recording, this preview must disappear. This can be done by invoking method *repaint* defined in class *LogoInterpreter* because the interpreter does not record the commands added to a macro in the history.