5. Exemplary Solutions: Seaside: Getting Started

Exercise 5.1, 5.2, 5.3

Trivial.

Exercise 5.4

```
WACounter >> decrease
    count := count + 2

WACounter >> decrease
    count := count - 3
```

Exercise 5.5, 5.6, 5.7, 5.8

Trivial.

Exercise 5.9

```
STUserNumberGuesser >> go
    | guess number numberOfGuesses |
    guess := -1. number := 100 atRandom.
    numberOfGuesses := 0.
    self inform: 'I am thinking of a number between 1 and 100.'.
    [ guess = number ] whileFalse: [
        guess := (self request: 'What is your guess?') asNumber.
        numberOfGuesses := numberOfGuesses + 1.
        guess < number
            ifTrue: [ self inform: 'The number I am thinking of
                      is bigger.' ].
        guess > number
            ifTrue: [ self inform: 'The number I am thinking of
                      is smaller.' ] ].
    self inform: 'You got it! Number of guesses needed: ',
                 numberOfGuesses asString
```

Exercise 5.10

The back button brings the application back to its previous state(s), as if the following actions (that is, guessing a number) would not have happened. This means that the number of guesses is increasing for following guesses starting with the value it has in the state to which you went back (CAUTION: This is dependent on your implementation of the number of guesses counter, see Ex. 12)

Exercise 5.11

Instance variables of any model object do not automatically get backtracked in Seaside, this means that if you have two views on the same session, both views access the same values of model instance variables. If you have modeled for instance the number of guesses as an instance variable, you will see the same number of guesses in both windows using the same session.

Exercise 5.12

With the back button you can easily cheat the application: Just guess the number until you got it right, go back to the first time you guessed the number and enter the correct number. The application will tell that you just had to guess once to get the number right.

Similarly, using the same session in different browser windows can, depending on your implementation, also be exploited to cheat in this application.

Exercise 5.13

One possible solution based on random guessing by the computer. Algorithmic ”guessing” is certainly also possible.

```
WATask subclass: #STComputerNumberGuesser
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Tutorial-Tasks'

STComputerNumberGuesser class >> initialize
    "self initialize"
    self registerAsApplication: 'tutorial/cng'

STComputerNumberGuesser >> go
    | guess numberOfGuesses done isTooBig min max |
    min := 1. max := 100.
    numberOfGuesses := 0.
    done := false.

    self inform: 'Think about a number between 1 and 100.'.
    [ done ] whileFalse: [
        numberOfGuesses := numberOfGuesses + 1.
        guess := (min to: max) atRandom.
        isTooBig := self confirm: 'Is ', guess asString, ' too big?'.
        isTooBig
            ifTrue: [max := guess - 1]
            ifFalse: [
                done := self confirm: 'Was the guess correct?'.
```

```
                done ifFalse: [min := guess + 1] ] ].
    self inform: 'Number of guesses the computer needed: ',
                        numberOfGuesses asString.
```

Exercise 5.14

The use of the #call: / #answer: mechanism is crucial in this exercise:

```
STChooseNumberGuesser >> go
    | userGuess numberOfGuesses |
    userGuess := self confirm: 'Do you want to guess a number yourself?'.
    numberOfGuesses := userGuess ifTrue: [self call: STUserNumberGuesser new]
                            ifFalse: [self call: STComputerNumberGuesser new].
    self inform: 'Number of required guesses: ',
                 numberOfGuesses asString.
```

.

In the other two #go methods, instead to #inform: about the number of guesses, answer them with:

```
    self answer: numberOfGuesses.
```

Exercise 5.15

```
STTicTacToeController class >> canBeRoot
    ^ true
```

.

Use the configuration web interface as stated to make the application accessible at http://localhost:8080/seaside/ttt.

Exercise 5.16

```
STTicTacToeController >> go
    self startGame.
    self playGame.
    self announceWinner.

STTicTacToeController >> startGame
    | userStarts |
    self newModel.
    userStarts := self confirm: 'Do you want to be the starting player
                                of the game?'.
    userStarts ifTrue: [self userMove].

STTicTacToeController >> playGame
    [self isFinished] whileFalse: [
        self computerMove.
```

```
        self isFinished ifFalse: [self userMove].
    ].

STTicTacToeController >> announceWinner
    self inform: 'The winner is: ', self nicelyDisplayedWinner.

STTicTacToeController >> nicelyDisplayedWinner
    ^self winner == -1
        ifTrue: ['The Computer']
        ifFalse: [
            self winner == 1
                ifTrue: ['You!']
                ifFalse: ['Nobody'] ]

STTicTacToeController >> isFinished
    ^self model isFinished

STTicTacToeController >> winner
    ^self model winner
```