Übungsgruppe 3; Zweiergruppe 23
274196 Lucas Brutschy & 273784 Philipp Fischer

Übung Programmierung

Blatt 6

Aufgabe 1 – Objektorientierte Programmierung und Datenabstraktion

a)

Ein Objekt ist eine Instanz einer Klasse. Die Klasse selbst gibt nur die Struktur eines Objektes an, dass später als Instanz dieser erzeugt werden kann. Die Klasse selbst kann keine Daten speichern, außer als `static` deklarierte.

b)

Die Techniken Call-By-Value und Call-By-Reference sind Arten Werte an eine Methode zu übergeben. Call-By-Reference übergibt nicht den Wert selbst, sondern nur eine Referenz, die auf den gespeicherten Wert zeigt. Call-By-Value übergibt eine Kopie des Wertes an die Methode. Ist der zu übergebende Wert eine Objektinstanz, wird bei Java nur die Referenz kopiert.

c)

Eine Prozedur ist in der Regel ein Unterprogramm ohne Rückgabewert, während eine Funktion nach ihrer Abarbeitung einen Wert zurückgibt. Eine Methode ist ein Unterprogramm welches sich auf ein bestimmtes Objekt, oder im Falle von statischen Methoden auf eine bestimmte Klasse bezieht. Ein solches Unterprogramm kann wie eine Funktion oder eine Prozedur verwendet werden.

d)

Attribute sind Teil eines Objekts. Sie existieren, solange das Objekt selber existiert. Lokale Variablen hingegen werden innerhalb von Methoden deklariert und existieren nur bis zum Ende ihres Blocks (z.B. der Schleife oder der Methode in der sie deklariert wurden).

e)

Ein ADT (abstract data type, Abstrakter Datentyp) ist ein Datentyp, der komplett unabhängig von der konkreten Implementierung spezifiziert wird. Im Allgemeinen setzt sich eine Spezifikation eines ADTs zusammen aus der Deklaration des Wertebereichs die ein solcher Datentyp annehmen kann, sowie einer Reihe von Operationen die auf diesen Datenyp angewendet werden können. Mit Java Interfaces kann eine Sonderform der ADTs angegeben werden, die die konkrete Implementierung auf ein Programm in Java einschränkt. Eine komplett abstrahierte Spezifikation von Datentypen ist in Java nicht möglich.

f)

Nichtstatische Klassenattribute sollten stets als `private` deklariert sein, damit man nicht direkt von außen auf sie zugreifen kann. Um dem Benutzer der Klasse diese Möglichkeit zu bieten, sollte man dann Selektoren verwenden, also

Übungsgruppe 3; Zweiergruppe 23
274196 Lucas Brutschy & 273784 Philipp Fischer

Methoden die das indirekte Setzen und Lesen der Attribute erlauben. Zweck eines solchen Klassenentwurfs ist es, die internen Vorgänge beim Lesen und Setzen der Attribute zu abstrahieren um eventuelle Abänderung dieser bei gleich bleibendem öffentlichem Inferface zu erlauben.

Aufgabe 2 – Programmanalyse

a)

Ausdruck 1

Methodensignatur	a	b
getInstance(double)		1
constructor a(int)	1.0	1
constructor a(double)	1.0	1
setA(int)	1.0	1
g (a)		1
getA()	10.0	1
getA()	10.0	1
setA(int)	1.0	1
constructor a(double)	1.0	1
setA (int)	1.0	1
setB(a)		1
getA()	100.0	1
f(a)		100
getInstance(a)	100.0	100
constructor a(a)	1.0	100
getA()	100.0	100
setA(int)	1.0	100
setB(a)		100
getA()	200.0	100

Ausdruck 2

Methodensignatur	a	b
getInstance(int)		200
constructor a(int)	1.0	200
constructor a(double)	1.0	200
setA(int)	1.0	200
g(a)		200
getA()	20.0	200
getA()	20.0	200
setA(int)	1.0	200
constructor a(double)	1.0	200
setA(int)	1.0	200
setB(a)		200
getA()	200.0	200

Übungsgruppe 3; Zweiergruppe 23
274196 Lucas Brutschy & 273784 Philipp Fischer

Ausdruck 3

Methodensignatur	a	b
getA()	400.0	200
g(int , dobule)	400.0	200
getInstance(int)		200
constructor a(int)	1.0	200
constructor a(double)	1.0	200
setA(int)	1.0	200
g(a)		200
getA()	4000.0	200
getA()	4000.0	200
setA(int)	1.0	200
constructor a(double)	1.0	200
setA(int)	1.0	200
setB(a)		200
getA()	40000.0	200
getA()	16000000.0	40000
getInstance(double)		40000
constructor a(int)	1.0	40000
constructor a(double)	1.0	40000
setA(int)	1.0	40000
g(a)		40000
getA()	10.0	40000
getA()	10.0	40000
setA(int)	1.0	40000
constructor a(double)	1.0	40000
setA(int)	1.0	40000
setB(a)		40000
getA()	100.0	40000
getA()	100.0	100

b)

Der Fehler befindet sich in der folgenden Methode:

```
public void g( double a, double b ) {
      setB( getInstance( a ).getA() * getInstance( b ).getA() );
}
```

In dieser Methode wird die Methode `setB(double)` aufgerufen, es existiert jedoch nur die Methode `setB(a)`, welche mit `getA()` den Wert der übergebenen Instanz von a abfragt und als `int` gecasted in b speichert.
Um den Fehler zu beheben fügen wir der Klasse die folgende, ergänzende Methode hinzu, welche auch direkt ein `double` als Argument annimmt:

```
public static void setB( double a ) {
      b = (int)a;
}
```

Aufgaben 3 und 4

Der Quelltext zu diesen Aufgaben befindet sich im Anhang.

Kasse.java

```java
/**
 * SHEET 6 - EXERCISE 3a
 * Multicurrency Cash System
 *
 * @author 273784 Philipp Fischer & 274196 Lucas Brutschy
 * @version 04.12.2006
 * platform: J2RE 1.5.0_08-b03, Linux 2.6.17-gentoo
*/
public class Kasse {
    // The euro-pound exchange rate is saved in that constant
    private static final float aEuroInPound = .677f;

    // This is the list of the items available to the customer
    private static final float[] itemEuroPrices = {
        149.90f,  //Item 00, standard adult ticket
        220.45f,  //Item 01, first clas adult ticket
        1340.00f, //Item 02, group ticket for up to 8 people
        30.0f,    //Item 03, child younger than or 6 years
        999.0f,   //Item 04, get Andreas K. sitting next to you

        1.00f,    //Item 05, extra drink
        2.50f,    //Item 06, vegetarian only food
        4.30f,    //Item 07, Menu 1
        4.50f     //Item 08, Menu 2
    };

    // This will be holding the total of the prices to pay
    // It will also be called 'internal sum' further on
    private float euroSum;

    /**
     * The constructor initializes the internal sum by invoking reset()
     */
    public Kasse() {
        reset();
    }

    /**
     * reset() sets the internal sum to zero
     */
    public void reset() {
        euroSum = 0;
    }

    /**
     * addItem adds the price of the given item to the list
     * @param itemID the item of wich to add the price
     */
    public void addItem(int itemID) {
        if(isItem(itemID)) {
            euroSum += getPrice(itemID);
        }
    }

    /**
     * subtractItem subtracts the price of a certain item from the list
     * @param itemID the item with the price to subtract
     */
    public void subtractItem(int itemID) {
        if(isItem(itemID)) {
            euroSum -= getPrice(itemID);
        }
    }

    /**
     * addVoucher adds the value of a voucher to the list
     * @param voucherEuroValue
     */
    public void addVoucher(float voucherEuroValue) {
        if(voucherEuroValue > 0) {
            euroSum -= voucherEuroValue;
        }
    }

    /**
     * getSum returns the current total to pay
     * @return the current total to pay
     */
    public float getSum() {
        return (euroSum<0?0:euroSum);
```

Kasse.java

```java
    }

    /**
     * getSum returns the current total to pay in pounds
     * @return the current total to pay in pounds
     */
    public float getSumInPounds() {
        return convertEuroToPounds(getSum());
    }

    /**
     * getPrice is used to retrieve the price of a certain item, if availabe
     * @param itemID the item of wich to get the price
     * @return the price of the item, if existent, 0 otherwise
     */
    public static float getPrice(int itemID) {
        return isItem(itemID)?itemEuroPrices[itemID]:0;
    }

    /**
     * isItem can be used to check if a certain item exists
     * @param itemID the item wich to check for existance
     * @return true if the item exists, false otherwise
     */
    public static boolean isItem(int itemID) {
        return (itemID >= 0 && itemID < itemEuroPrices.length);
    }

    /**
     * This method converts a euro-value to pounds
     * @param euroPrice the price to convert (in euro)
     * @return the converted price in pounds
     */
    public static float convertEuroToPounds(float euroPrice) {
        return euroPrice * aEuroInPound;
    }

    /**
     * This method converts a pound-value to euros
     * @param poundPrice the price to convert (in pounds)
     * @return the converted price in euro
     */
    public static float convertPoundsToEuro(float poundPrice) {
        return poundPrice / aEuroInPound;
    }

}
```

KasseTestDrive.java

```java
/**
 * SHEET 6 - EXERCISE 3b
 * Kasse-class Testing System
 *
 * @author 273784 Philipp Fischer & 274196 Lucas Brutschy
 * @version 04.12.2006
 * platform: J2RE 1.5.0_08-b03, Linux 2.6.17-gentoo
*/
public class KasseTestDrive {
    public static void main (String[] args) {
        System.out.println("=== Kasse ===");

        Kasse testKasse = new Kasse();
        testKasse.addItem(1);
        testKasse.addItem(4);
        testKasse.addItem(8);
        testKasse.addItem(10);

        testKasse.addVoucher(Kasse.convertPoundsToEuro(10));

        System.out.println("Total: " + testKasse.getSum());

    }
}

```

Stack.java

```java
/**
 * SHEET 6 - EXERCISE 4a
 * Abstract Stack class implementation
 *
 * @author 273784 Philipp Fischer & 274196 Lucas Brutschy
 * @version 04.12.2006
 * platform: J2RE 1.5.0_08-b03, Linux 2.6.17-gentoo
*/
public class Stack {
    private char[] currentArray;
    private final int initialSize,increment,decrement;
    private int numberOfItems;

    /**
     * This is the actual constructor of the class
     * It initializes the properties "increment", "decrement"
     * ans "initialSize" and clears (and thereby initializes)
     * the stack
     * @param initialSize The number of elements to allocate space for at the beginning
     * @param increment The incrementation threshold
     * @param decrement The decrementation threshold
     */
    public Stack(int initialSize, int increment, int decrement) {
        this.increment = Math.max(increment, 1);
        this.decrement = Math.max(decrement, 1);
        this.initialSize = Math.max(initialSize, 1);
        clear();
    }

    /**
     * This alternate constructor initializes both of the threshold values to 5
     * and lets the user only specify the initialSize. It references to the
     * topmost constructor
     * @param initialSize
     */
    public Stack(int initialSize) {
        this(initialSize, 5, 5);
    }

    /**
     * This is the standard constructor which will set the threshold values to
     * 5 and start with an initial size of 4 elements
     */
    public Stack() {
        this(4, 5, 5);
    }

    /**
     * This method creates a new stack with no items in it, overwrites the old
     * array with it and thereby clears anything that was stored on this stack before
     */
    public void clear() {
        currentArray = new char[initialSize];
        numberOfItems = 0;
    }

    /**
     * This is a private helper method, that automates the process of resizing the
     * internal array. It creates a new array of the new size wanted and copies
     * all elements it can copy from the old array to the new one.
     * @param oldSize The size of the old internal array
     * @param newSize The size the new array should have
     */
    private void resizeStack(int oldSize, int newSize) {
        newSize = Math.max(newSize, 1);
        char[] newStack = new char[newSize];
        System.arraycopy(currentArray, 0, newStack, 0, Math.min(newSize, oldSize));
        currentArray = newStack;
    }

    /**
     * This method pushes a character onto the stack. If necessary it increases
     * the size of the internal array.
     * @param newChar The character to add to the stack
     */
    public void push (char newChar) {
        if(++numberOfItems>currentArray.length) {
            resizeStack(currentArray.length, currentArray.length + increment);
        }
        currentArray[numberOfItems-1] = newChar;
```

Stack.java

```
81        }
82
83        /**
84         * This method reads a character from the stack und decreases the number
85         * of items by one. If hereby the size falls below the decrement threshold,
86         * the array is being resized.
87         * @return The character on top of the stack
88         */
89        public char pop () {
90            if(isEmpty()) { return '\0'; }
91            char charToReturn = currentArray[--numberOfItems];
92            if(numberOfItems <= currentArray.length - decrement) {
93                resizeStack(currentArray.length, currentArray.length - decrement);
94            }
95            return charToReturn;
96        }
97
98        /**
99         * Checks if the array is empty
100        * @return True if it is empty, fals otherwise
101        */
102       public boolean isEmpty () {
103           return (numberOfItems == 0);
104       }
105   }
106
```

InfixToPostfix.java

```java
/**
 * SHEET 6 - EXERCISE 4b
 * Infix to Postfix Notation Convertor
 * with error and warning system
 *
 * @author 273784 Philipp Fischer & 274196 Lucas Brutschy
 * @version 04.12.2006
 * platform: J2RE 1.5.0_08-b03, Linux 2.6.17-gentoo
*/
public class InfixToPostfix {;
    /**
     * This is only used as a shortcut to System.out.println()
     * @param line the line to print to the console
     */
    private static void print(String line) {
        System.out.println(line);
    }

    /**
     * This method prints out an error to the console and shows the position
     * it occured at to the user
     * @param position the position of the error
     * @param errorString the message to display to the user
     */
    private static void quitWithErrorAt(int position, String errorString) {
        for(int i = 0; i < position; i++) {
            System.out.print(" ");
        }
        print("^");
        print("Error at position "+position+": "+errorString);
        System.exit(0);
    }

    /**
     * This method prints out a warning to the console and shows the position
     * it occured at to the user
     * @param position the position of the character wich the warning is related to
     * @param errorString the message to display to the user
     */
    private static void warnAt(int position, String errorString) {
        for(int i = 0; i < position; i++) {
            System.out.print(" ");
        }
        print("^");
        print("Warning at position "+position+": "+errorString);
    }

    /**
     * This is the entry point of the program
     * @param args unused
     */
    public static void main (String[] args) {
        print("=== Infix to Postifx convertor ===");

        // If the user did not pass exactly one argument, quit the program with this message
        if(args.length != 1) {
            print("You must enter exactly 1 argument as infix term.");
            return;
        }

        /*
         * You may uncomment some of the following examples to try out the error system
         */

        //=== SIMPLE EXAMPLES ===
        //args[0] = "(5 * 6)";
        //args[0] = "(5 * (6 + 7))";
        //args[0] = "(5 * (((9 + 8) * (4 * 6)) + 7))";

        //=== ERROR EXAMPLES ===
        //-- "Two operators in one scope" example --
        //args[0] = "5 * (((9 + 8 * (4 * 6)) + 7)";

        //-- "Operator missing"-error example --
        //args[0] = "(9 + 8) (4 * 5)";

        //-- Another "Operator missing"-error example --
        //args[0] = "(9 + 8) * (4 5)";

        //-- "Misplaced Operator" example --
        //args[0] = "( 5 * + 4 )";

        //-- "non-matching bracket" example --
        //args[0] = "((9 + (4 * 6)) + 7))";

        //-- "invalid character" example --
        //args[0] = "(5 / 4)";


```

InfixToPostfix.java

```
90          //=== WARNING EXAMPLES ===
91          //-- Another more complex "Operator missing" example --
92          //args[0] = "( 5 * ((9+8)) )";
93
94          //-- "Operator missing"-warning example --
95          //args[0] = "(9 + 8) * (4 *  (5))";
96
97          //-- "Unclosed bracket"-warning example --
98          //args[0] = "((9 + 8) * (4 * 5)";
99
100
101         // Print out the term the user entered.
102         // Later on it will be used to show where the errors occured
103         print("You entered: ");
104         print(args[0]);
105
106         // We convert the term-string to a character-array
107         char[] term = args[0].toCharArray();
108
109         // The string 'result' will be used to construct the postfix-notation term
110         String result = "";
111
112         // This integer is used to store the current bracket-nesting-depth
113         int depth = 0;
114
115         // Saves if an operator or a closing bracket is expected as the next character
116         boolean expectingOperatorOrScopeEnd = false;
117
118         // Here the operators of the different depth-levels will be saved
119         Stack operators = new Stack();
120
121         /* In this additional Stack we store the depth for each of the operators in
122          * the operators Stack.
123          * This is needed for the error system to see, if an operator occured twice
124          * at a certain depth-level
125          */
126         Stack operatorDepth = new Stack();
127
128         // This for-loop walks through each character of the infix-term
129         for(int i = 0; i < term.length; i++) {
130             // We can safely ignore any spaces
131             if(term[i] == ' ') continue;
132
133             // CASE: The current character is an operator
134             if(term[i] == '+' || term[i] == '*') {
135                 // If we are not expecting an operator, it must be misplaced
136                 if(!expectingOperatorOrScopeEnd) {
137                     quitWithErrorAt(i, "Misplaced operator!");
138                 }
139
140                 /* If we have saved any operators on the stack, we will
141                  * get the depth of the operator that is on top of the stack and
142                  * check if this is the same depth we are currently in.
143                  * If this is the case, it must be the second operator in
144                  * the current scope. We will print out an error.
145                  * If not, push back the operator-depth to its stack.
146                  */
147                 if(!operatorDepth.isEmpty()) {
148                     char lastOperatorsDepth = operatorDepth.pop();
149                     if(lastOperatorsDepth == depth) {
150                         quitWithErrorAt(i, "This is the second operator in that scope (Only one is allowed)!");
151                     } else {
152                         operatorDepth.push(lastOperatorsDepth);
153                     }
154                 }
155
156                 // Store the new operator and its depth in their stacks respectively
157                 operators.push(term[i]);
158                 operatorDepth.push((char)depth);
159
160                 // Now we are not expecting an operator anymore
161                 expectingOperatorOrScopeEnd = false;
162
163             // CASE: The current character is a number
164             } else if(term[i] >= '0' && term[i] <= '9') {
165                 // If we were actually expecting an operator here, complain
166                 if(expectingOperatorOrScopeEnd) {
167                     quitWithErrorAt(i, "Operator is missing before that digit!");
168                 }
169
170                 // Add the digit to the end of our final postfix-term
171                 result += term[i] + " ";
172
173                 // Now we are expecting an operator or the end of the scope.
174                 expectingOperatorOrScopeEnd = true;
175
176             // CASE: The current character opens a scope
177             } else if(term[i] == '(') {
178                 /* We can only allow a maximum depth of 32768, because
```

InfixToPostfix.java

```
                 * the depths are saved in a char-Stack wich can only
                 * hold values up to 32768
                 */
                if(depth == 32768) {
                    quitWithErrorAt(i, "A maximum depth of 32768 is allowed!");
                }

                // If an operator was expected but a new scope is starting, throw an error
                if(expectingOperatorOrScopeEnd) {
                    quitWithErrorAt(i, "Operator is missing before that scope!");
                }

                // Increase our current depth-level
                depth++;

            // CASE: The current character closes a scope
            } else if(term[i] == ')') {
                // Throw an error, if we are not expecting a closing bracket, e.g. (4 + )
                if(!expectingOperatorOrScopeEnd) {
                    quitWithErrorAt(i, "Not expecting the end of the scope here!");
                }

                // If we are at root depth but still reach a closing bracket, throw an error
                if(depth <= 0) {
                    quitWithErrorAt(i, "Bracket does not match!");
                }


                /* Here we check the depth-level of the last operator in the stack
                 * against the level of the scope being closed now.
                 * If they don't match, there wasn't any operator in that scope, and we'll ignore it.
                 * If there did occur one, we will now add it to the postfix-term.
                 */
                char lastOperatorsDepth = operatorDepth.pop();
                if(lastOperatorsDepth != depth) {
                    warnAt(i, "Operator is missing in that scope!");
                    operatorDepth.push(lastOperatorsDepth);
                } else {
                    result += operators.pop() + " ";
                }

                // Decrease the current depth-level
                depth--;

                // After that scope, there should be an operator or another ending scope
                expectingOperatorOrScopeEnd = true;

            // CASE: If none of the above characters match with the current, it must be invalid
            } else {
                quitWithErrorAt(i, "Invalid character not of 0123456789+*()");
            }
        }

        /* If now, at the and of the infix-term, we still are at non-root-level]
         * there must be unclosed brackets left. Warn the user about that.
         */
        if(depth > 0) {
            print("Warning: Unclosed brackets remain.");
        }

        /* We will pop off any remaining operators from the stack and append them to
         * the postfix-term. This is the case if either unclosed brackets remain, or
         * an operator was placed at root-level of the infix-term
         */
        while(!operators.isEmpty()) {
            result += operators.pop() + " ";
        }

        // Finally we can print the postfix converted term out to the console
        print("In postfix-notation that is:");
        print(result);

    }
}
```