Question 1: MyOwnString

a) The idea of my solution

We create the class MyOwnString with two private fields: an array of chars “data” and an integer number “length”. For easy creation of an empty string we create a private method which will assign to the length zero value and assign null value to the data array. Then we make four constructors. The constructor without any parameters will create the empty string, the constructor with an array of chars as an input parameter will copy them to the our data if the reference is not equal to null, the third constructor with a String type parameter will copy characters from it, and the last one constructor will work in the same way as the second constructor.

In the method “compareTo” first we check is an input parameter is equal to null and if yes, then return 1, otherwise assign to a result value 0 and gathering only letters from both MyOwnString to String type variables making them lower case. After this in a loop while we compare each letter one by one until we find different ones or reach minimal length of the two created strings. If all characters were the same, then we compare the lengths and return a correspondent result.

In the method “concat” first we check is an input parameter is equal to null or an empty string and if yes, then do nothing. If the current string is empty, just copy the input data to the current, otherwise initialize a new array with a correspondent size, copy characters from the both data arrays and assign a new length value. The static method “concat” just initialize a new MyOwnString object with the input parameter in the constructor as the first string, call the method “concat” described above with the second parameter and return the initialized MyOwnString.

In the method “indexOf” first we check is an input parameter is equal to null and if yes, then return -1. If the input parameter is an empty string, then return 0 as a first accuracy in the current string. Otherwise go through the current array and check if the input string exists in it as a substring using counter of equal characters. If we found the counter that equals to size of the input string, return an index of the first equal character, otherwise -1.

The method “length” just returns value of the field “length”.

In the method “substring” first we check for correctness input values. If they are incorrect, return null. Otherwise copy a range of characters that we need to a new array with the correspondent size and return a new MyOwnString variable with the created array as an input parameter in the constructor of it.

In the method “toString” using the String.copyValueOf(char[] data) method we can easily return the result.

In the method “trim” first we count white spaces at the beginning of the string. If this value is equal to the length of the string, make this string empty and return its previous lengths. Otherwise count white spaces from the end of the string. Eventually copy all characters between this two values to a new array, copy a reference of it to the our current array, its length and return a sum of the previously calculated counters of white spaces.

b) Source code

- MyOwnString.java

```
package uebung7.question1;

import java.util.Arrays;

/** The class MyOwnString represents a small copy of the already created class
 * String and implements some methods
 *
 * @author Andrii Dzhyrma */
public class MyOwnString {

  // Static Members ------------------------------------------------------------

  /** Adds the string s2 to the end of s1 and returns the result as MyOwnString
   *
   * @param s1 - the MyOwnString to work with
   * @param s2 - the MyOwnString to be concatenated
   * @return a result of concatenation */
  public static MyOwnString concat(MyOwnString s1, MyOwnString s2) {
    MyOwnString result = new MyOwnString(s1);
    result.concat(s2);
    return result;
  }

  // Fields --------------------------------------------------------------------

  private char[] data;
  private int length;

  // Private Methods -----------------------------------------------------------

  /* Creates an empty string */
  private void createEmptyString() {
    data = null;
    length = 0;
  }

  // Constructors --------------------------------------------------------------

  /** This constructor creates an empty string */
  public MyOwnString() {
    createEmptyString();
  }

  /** This constructor copies the content of rawData and sets the length
   * depending on the number of characters in the existing array */
  public MyOwnString(char[] rawData) {
    if (rawData == null || rawData.length == 0) {
      createEmptyString();
      return;
    }
    // copy an input data using Arrays class2
    data = Arrays.copyOf(rawData, rawData.length);
    length = data.length;
  }

  /** This constructor creates a new string initialized with the content of
   * MyOwnString s */
  public MyOwnString(MyOwnString s) {
    if (s == null || s.length == 0 || s.data == null) {
      createEmptyString();
      return;
    }
    data = Arrays.copyOf(s.data, s.data.length);
    length = s.length;
  }

  /** This constructor initializes the instance with characters of the string s */
  public MyOwnString(String s) {
    if (s == null || s.length() == 0) {
      createEmptyString();
      return;
    }
    data = s.toCharArray();
    length = data.length;
```

```
}

// Public Methods ------------------------------------------------------------

/** Compares the string to s lexicographically and returns a negative integer
 * if this string object lexicographically precedes the argument s, a positive
 * integer if this string object lexicographically follows the argument s, and
 * zero if the two given strings are equal <br>
 * <b>Note: thereby only letter character A-Z and a-z shall be considered for
 * the comparison; any other character will be ignored</b>
 *
 * @param s - the MyOwnString to compare with
 * @return <b>-1</b> - if this string object lexicographically precedes the
 *         argument s<br>
 *         <b>0</b> - if the two given strings are equal<br>
 *         <b>1</b> - if this string object lexicographically follows the
 *         argument s */
public int compareTo(MyOwnString s) {
  // if an input string is null, return 1
  if (s == null) return 1;
  int result = 0;
  // create two strings without any additional symbols except letter
  // characters and make them lower case
  String s1 = "", s2 = "";
  if (length != 0 && data != null) {
    for (int i = 0; i < length; i++)
      if ((data[i] >= 'A' && data[i] <= 'Z')
          || (data[i] >= 'a' && data[i] <= 'z'))
        s1 += Character.toLowerCase(data[i]);
  }
  if (s.length != 0 && s.data != null) {
    for (int i = 0; i < s.length; i++)
      if ((s.data[i] >= 'A' && s.data[i] <= 'Z')
          || (s.data[i] >= 'a' && s.data[i] <= 'z'))
        s2 += Character.toLowerCase(s.data[i]);
  }
  // calculate a minimal length
  int minLength = (s1.length() < s2.length()) ? s1.length() : s2.length();
  int i = 0;
  // continue while symbols are equal and we did not reach the minimal length
  while (result == 0 && i < minLength) {
    char currentValue = s1.charAt(i);
    char paraValue = s2.charAt(i);
    // compare them
    if (currentValue != paraValue)
      result = (currentValue < paraValue) ? -1 : 1;

    i++;
  }
  // if the result is still equal to 0, than compare lengths of strings
  if (result == 0)
    if (s1.length() != s2.length())
      result = (s1.length() < s2.length()) ? -1 : 1;
  return result;
}

/** Adds the string s to the end of the existing string
 *
 * @param s - the MyOwnString to be added */
public void concat(MyOwnString s) {
  // if an input data is and empty string or null, than do nothing
  if (s == null || s.length == 0 || s.data == null) return;
  // if the current string is empty, than copy the input string
  if (length == 0) {
    data = Arrays.copyOf(s.data, s.data.length);
    length = s.length;
    return;
  }
  // otherwise making a new array and concatenate two arrays in it
  char[] newData = Arrays.copyOf(data, length + s.length);
  for (int i = 0; i < s.length; i++)
    newData[length + i] = s.data[i];
  data = newData;
  length += s.length;
}

/** Returns the index within this string of the first occurrence of the
 * specified string argument s, returns a negative value if the substring was not found
```

```
     *
     * @param s - specified MyOwnString argument
     * @return a negative value if the substring was not found or a position of
     *         the first occurrence */
    public int indexOf(MyOwnString s) {
      // if an input string is null, than return -1
      if (s == null) return -1;
      // if the input string is empty, than return 0
      if (s.length == 0 || s.data == null) return 0;
      // otherwise try to find a fist occurrence and return its position if found
      for (int i = 0; i < length - s.length + 1; i++) {
        int index = 0;
        for (; index < s.length && data[index + i] == s.data[index]; index++)
          ;
        if (index == s.length) return i;
      }
      // return -1 if we did not find any position
      return -1;
    }

    /** Returns the length of the string */
    public int length() {
      return length;
    }

    /** Returns the string starting at the character with index start (inclusive)
     * to the character with index end (exclusive); returns null if any of the
     * indexes given is not valid for the string
     *
     * @param start - the start index
     * @param end - the end index
     * @return a substring from the start index to the end index */
    public MyOwnString substring(int start, int end) {
      // if an input data is incorrect, return null
      if (start > end || start < 0 || end > length) return null;
      // if the start index is equal to the end index, return a new empty
      // MyOwnString
      if (start == end) return new MyOwnString();
      // otherwise it is a range between the start and the end indices
      char[] substring = Arrays.copyOfRange(data, start, end);
      return new MyOwnString(substring);
    }

    @Override
    /** Returns the saved string as an instance of java.lang.String */
    public String toString() {
      return (data == null) ? "" : String.copyValueOf(data);
    }

    /** Eliminate all leading and tailing white-spaces of the string
     *
     * @return a number of white-spaces eliminated */
    public int trim() {
      // if the current string is empty, than nothing to trim
      if (length == 0 || data == null) return 0;
      // calculate a number of white spaces at the beginning
      int startIndex = 0;
      while (startIndex < length && data[startIndex] == ' ')
        startIndex++;
      // if this number is equal to the length, make an empty string
      if (startIndex == length) {
        createEmptyString();
        return startIndex;
      }
      // calculate a number of white spaces at the end
      int endIndex = length - 1;
      while (data[endIndex] == ' ')
        endIndex--;
      // calculate a new length and return 0 if there is nothing to trim
      int newLength = endIndex - startIndex + 1;
      if (newLength == length) return 0;
      // copy a reference of the trimmed range of data
      int result = length - newLength;
      data = Arrays.copyOfRange(data, startIndex, endIndex + 1);
      length = newLength;
      return result;
    }
}
```

- Main.java

```
package uebung7.question1;

public class Main {
  private static final String OUTPUT_FORMAT_STRING = "%s = \"%s\", length = %d%n";

  /** @param args - no arguments will evaluate */
  public static void main(String[] args) {
    System.out.println("\nCostructors----------------------------------------");
    String s = null;
    MyOwnString nullString = new MyOwnString(s);
    System.out.printf(OUTPUT_FORMAT_STRING, "nullString", nullString, nullString.length());
    MyOwnString empty1 = new MyOwnString();
    System.out.printf(OUTPUT_FORMAT_STRING, "empty1", empty1, empty1.length());
    MyOwnString empty2 = new MyOwnString("");
    System.out.printf(OUTPUT_FORMAT_STRING, "empty2", empty2, empty2.length());
    MyOwnString empty3 = new MyOwnString(new char[] {});
    System.out.printf(OUTPUT_FORMAT_STRING, "empty3", empty3, empty3.length());
    MyOwnString empty4 = new MyOwnString(empty1);
    System.out.printf(OUTPUT_FORMAT_STRING, "empty4", empty4, empty4.length());
    MyOwnString hello = new MyOwnString(new char[] { 'H', 'e', 'l', 'l', 'o' });
    System.out.printf(OUTPUT_FORMAT_STRING, "hello", hello, hello.length());
    MyOwnString world = new MyOwnString(" World!");
    System.out.printf(OUTPUT_FORMAT_STRING, "world", world, world.length());
    System.out.println("\nConcat----------------------------------------------");
    MyOwnString helloWorld = MyOwnString.concat(hello, world);
    System.out.printf(OUTPUT_FORMAT_STRING, "helloWorld", helloWorld,
        helloWorld.length());
    MyOwnString concat = new MyOwnString("Example");
    System.out.printf(OUTPUT_FORMAT_STRING, "concat", concat, concat.length());
    concat.concat(null);
    System.out.printf("Null concatenated:%n" + OUTPUT_FORMAT_STRING,
        "concat", concat, concat.length());
    concat.concat(empty1);
    System.out.printf("Empty string concatenated:%n" + OUTPUT_FORMAT_STRING,
        "concat", concat, concat.length());
    empty1.concat(concat);
    System.out.printf(
        "To the empty string concatenated the string \"concat\":%n"
            + OUTPUT_FORMAT_STRING, "empty1", empty1, empty1.length());
    MyOwnString whiteSpaces = new MyOwnString("    ");
    System.out.printf(OUTPUT_FORMAT_STRING, "whiteSpaces", whiteSpaces,
        whiteSpaces.length());
    MyOwnString stringToTrim = MyOwnString.concat(whiteSpaces, concat);
    stringToTrim.concat(whiteSpaces);
    System.out.printf(OUTPUT_FORMAT_STRING, "stringToTrim", stringToTrim,
        stringToTrim.length());
    System.out.println("\nTrim------------------------------------------------");
    int trimmed = stringToTrim.trim();
    System.out.println("stringToTrim first trimming:");
    System.out.printf("trimmed = \"%d\", " + OUTPUT_FORMAT_STRING, trimmed,
        "stringToTrim", stringToTrim, stringToTrim.length());
    trimmed = stringToTrim.trim();
    System.out.println("stringToTrim second trimming:");
    System.out.printf("trimmed = \"%d\", " + OUTPUT_FORMAT_STRING, trimmed,
        "stringToTrim", stringToTrim, stringToTrim.length());
    trimmed = whiteSpaces.trim();
    System.out.println("whiteSpaces first trimming:");
    System.out.printf("trimmed = \"%d\", " + OUTPUT_FORMAT_STRING, trimmed,
        "whiteSpaces", whiteSpaces, whiteSpaces.length());
    trimmed = whiteSpaces.trim();
    System.out.println("whiteSpaces second trimming:");
    System.out.printf("trimmed = \"%d\", " + OUTPUT_FORMAT_STRING, trimmed,
        "whiteSpaces", whiteSpaces, whiteSpaces.length());
    System.out.println("\nSubstring-------------------------------------------");
    System.out.printf(OUTPUT_FORMAT_STRING, "helloWorld.substring(0, 12)",
        helloWorld.substring(0, 12), helloWorld.substring(0, 12).length());
    System.out.printf(OUTPUT_FORMAT_STRING, "helloWorld.substring(2, 10)",
        helloWorld.substring(2, 10), helloWorld.substring(2, 10).length());
    System.out.printf(OUTPUT_FORMAT_STRING, "helloWorld.substring(0, 0)",
        helloWorld.substring(0, 0), helloWorld.substring(0, 0).length());
    System.out.printf(OUTPUT_FORMAT_STRING, "helloWorld.substring(0, 20)",
        helloWorld.substring(0, 20), 0); // returns null
    System.out.printf(OUTPUT_FORMAT_STRING, "helloWorld.substring(-5, 12)",
        helloWorld.substring(-5, 12), 0); // returns null
    System.out.printf(OUTPUT_FORMAT_STRING, "helloWorld.substring(12, 0)",
```

```
            helloWorld.substring(12, 0), 0); // returns null
        System.out.println("\nCompareTo----------------------------------------");
        System.out
        .println("empty2.compareTo(null) = " + empty2.compareTo(null));
        System.out
            .println("empty2.compareTo(empty2) = " + empty2.compareTo(empty2));
        System.out
            .println("empty2.compareTo(empty3) = " + empty2.compareTo(empty3));
        System.out.println("hello.compareTo(hello) = " + hello.compareTo(hello));
        System.out.println("hello.compareTo(world) = " + hello.compareTo(world));
        System.out.println("world.compareTo(hello) = " + world.compareTo(hello));
        System.out.println("hello.compareTo(helloWorld) = "
            + hello.compareTo(helloWorld));
        System.out
            .println("MyOwnString.concat(new MyOwnString(\"   \"), hello).compareTo(hello) = "
                + MyOwnString.concat(new MyOwnString("   "), hello)
                    .compareTo(hello));
        System.out
            .println("MyOwnString.concat(hello, new MyOwnString(\"   \")).compareTo(hello)) = "
                + MyOwnString.concat(hello, new MyOwnString("   "))
                    .compareTo(hello));
        System.out.println("\nIndexOf----------------------------------------");
        System.out.println("helloWorld.indexOf(null) = "
            + helloWorld.indexOf(null));
        System.out.println("helloWorld.indexOf(empty2) = "
            + helloWorld.indexOf(empty2));
        System.out.println("helloWorld.indexOf(hello) = "
            + helloWorld.indexOf(hello));
        System.out.println("helloWorld.indexOf(world) = "
            + helloWorld.indexOf(world));
        System.out.println("helloWorld.indexOf(new MyOwnString(\"o\")) = "
            + helloWorld.indexOf(new MyOwnString("o")));
        System.out.println("helloWorld.indexOf(new MyOwnString(\"a\")) = "
            + helloWorld.indexOf(new MyOwnString("a")));
        System.out.println("empty2.indexOf(empty2) = " + empty2.indexOf(empty2));
        System.out.println("empty2.indexOf(helloWorld) = "
            + empty2.indexOf(helloWorld));
    }
}
```

a) Test plan

#	Description	Expected output
1	Test all constructors with null, empty and common values	**A correspondent reaction as described in the requirements**
2	Test the methods “concat” with null, empty and common values	**A correspondent reaction as described in the requirements**
3	Test the method “trim” with empty and common values	**A correspondent reaction as described in the requirements**
4	Test the method “substring” with incorrect and common values	**A correspondent reaction as described in the requirements**
5	Test the method “compareTo” with null, empty and common values	**A correspondent reaction as described in the requirements**
6	Test the method “indexOf” with null, empty and common values	**A correspondent reaction as described in the requirements**

b) The output of the program

```
Costructors----------------------------------------
nullString = "", length = 0
empty1 = "", length = 0
empty2 = "", length = 0
empty3 = "", length = 0
empty4 = "", length = 0
hello = "Hello", length = 5
world = " World!", length = 7

Concat---------------------------------------------
helloWorld = "Hello World!", length = 12
concat = "Example", length = 7
Null concatenated:
concat = "Example", length = 7
Empty string concatenated:
concat = "Example", length = 7
To the empty string concatenated the string "concat":
empty1 = "Example", length = 7
whiteSpaces = "    ", length = 4
stringToTrim = "    Example    ", length = 15

Trim-----------------------------------------------
stringToTrim first trimming:
trimmed = "8", stringToTrim = "Example", length = 7
stringToTrim second trimming:
trimmed = "0", stringToTrim = "Example", length = 7
whiteSpaces first trimming:
trimmed = "4", whiteSpaces = "", length = 0
whiteSpaces second trimming:
trimmed = "0", whiteSpaces = "", length = 0

Substring------------------------------------------
helloWorld.substring(0, 12) = "Hello World!", length = 12
helloWorld.substring(2, 10) = "llo Worl", length = 8
helloWorld.substring(0, 0) = "", length = 0
helloWorld.substring(0, 20) = "null", length = 0
helloWorld.substring(-5, 12) = "null", length = 0
helloWorld.substring(12, 0) = "null", length = 0

CompareTo------------------------------------------
empty2.compareTo(null) = 1
empty2.compareTo(empty2) = 0
empty2.compareTo(empty3) = 0
hello.compareTo(hello) = 0
hello.compareTo(world) = -1
world.compareTo(hello) = 1
hello.compareTo(helloWorld) = -1
MyOwnString.concat(new MyOwnString("   "), hello).compareTo(hello) = 0
MyOwnString.concat(hello, new MyOwnString("   ")).compareTo(hello)) = 0

IndexOf--------------------------------------------
helloWorld.indexOf(null) = -1
helloWorld.indexOf(empty2) = 0
helloWorld.indexOf(hello) = 0
helloWorld.indexOf(world) = 5
helloWorld.indexOf(new MyOwnString("o")) = 4
helloWorld.indexOf(new MyOwnString("a")) = -1
empty2.indexOf(empty2) = 0
empty2.indexOf(helloWorld) = -1
```

Question 2: ToDo-Entries

a) The idea of my solution

We create a List class with two important constants (maximal number for an array of groups and an array of items), an static field “currentId” which will store last used id for initialization, the two array of groups and items, three integer variables (sizes of the arrays and the id of the current object) and methods. A constructor generates only the id for the current object. The “insertGroup” method first checks whether the current size of the array is less than maximal, checks an input group for correctness and if it is, finds correspondent place in the array

due to the order described in the requirements. During the searching process we check whether this group has been already added to the array or not. If not we shift all the elements on the right side by one place to the right and insert the input group in the found place. The “insertItem” works almost the same with some differences: we try to find place in the array due to another order and insert the item also to the group using insert(Item item) method. The “printAll” method prints an error if there are no elements in the array of items and prints all existing elements otherwise. The “printInGroups” method prints an error if there are no elements in the array of groups and prints all existing groups otherwise.

We create a Group class with a one important constant (maximal number for an array of items), a static field “currentId” which will store last used id for initialization, an array of items, four integer variables (size of the array, sorting order, unique id and a priority number), a name of the group and methods. A constructor generates the unique id, sets the name and the priority from input parameters. The “insert” method checks whether the array of items is full and checks an input item for correctness. If the item is correct and the array is not full, checks if it is not in the array and adds it after the last element. Then using the switch statement sort the array in a correspondent order. Additionally we create two methods “isEmpty” and “isFull” to check whether the array of items is empty or full. The “printAll” method prints a header for the group and all existing elements. The last two method “sortByDueDate” and “sortByStatus” sort the array of items using the algorithm of finding the minimal element and stores to the sort order variable a new value.

We create an Item class with a static field “currentId” which will store last used id for initialization, two integer values the id and the status, a description about the item, a date of the event, a variable of the parent group and methods. A constructor generates the unique id, sets the description, date and the current status. Additionally we create two methods “finish” and “start” which change status on “finished” or “started” correspondently. The “print” method prints the current item in the format as was described in the requirements. For the status, date and the description we create getters and only for the parent group we create the getter and the setter.

b) Source code

- List.java

```
package uebung7.question2;

/** The class List allows to store and maintain groups and items.
 *
 * @author Andrii Dzhyrma */
public class List {

  // Private Members ////////////////////////////////////////////////////////

  // Constants ---------------------------------------------------------------
  private static final String PRINT_FORMAT_STRING = "%nList #%d";
  private static final String NO_ITEMS_ERRORS_STRING = "There are no items in this list.";
  private static final String NO_GROUPS_ERRORS_STRING = "There are no groups in this list.";
  private static final int MAX_GROUPS = 4;
  private static final int MAX_ITEMS = 6;

  // Static Fields -----------------------------------------------------------
  private static int currentId = 1;

  // Fields ------------------------------------------------------------------
  private final Group[] groups = new Group[MAX_GROUPS];
  private final Item[] items = new Item[MAX_ITEMS];
  private int groupsSize = 0;
  private int itemsSize = 0;
  private int id;
```

```
// Public Members ////////////////////////////////////////////////////////

// Constructors -----------------------------------------------------------
/** This constructor initializes and generates id for the object. */
public List() {
  id = currentId++;
}

// Methods ----------------------------------------------------------------
/** Inserts a group into the array named groups of a list if not already
 * present in the list. Groups will be sorted according to their priority
 * (starting with 1) and their name.
 *
 * @param group - the group to be added
 * @return In case no insertion can be made false should be returned,
 *         otherwise true. */
public boolean insertGroup(Group group) {
  // if a reference of the group is equal to null, or it has some items in the
  // array (means that this grouped already used in an other list), or the
  // size of the current array of groups is maximal, then return false.
  if (group == null || !group.isEmpty() || groupsSize == MAX_GROUPS)
    return false;
  // insert the group into the correspondent place in the array to keep it
  // sorted
  int index = 0;
  for (; index < groupsSize
      && (groups[index].getPriority() < group.getPriority() || (groups[index]
          .getPriority() == group.getPriority() && groups[index].getName()
          .compareToIgnoreCase(group.getName()) <= 0)); index++)
    // return false if this group already exists
    if (groups[index] == group)
      return false;
  // shift groups from the right side by one
  for (int i = groupsSize++; i > index; i--)
    groups[i] = groups[i - 1];
  // insert the input group to the space created
  groups[index] = group;
  return true;
}

/** Inserts an item into the list and into the according existing group within
 * the list if and only if the item is not yet contained within the list and
 * the specified group exists. The items of a list will be ordered by due
 * date, then status and then name.
 *
 * @param item - the item to be added
 * @param group - the group in which item should be inserted
 * @return In case no insertion can be made false should be returned,
 *         otherwise true. */
public boolean insertItem(Item item, Group group) {
  // if a reference of the input item is equal to null, or the item already
  // belongs to some group, or a reference to the group is equal to null, or
  // the group is full, or the current array of items is full, than return
  // false
  if (item == null || item.getBelongTo() != null || group == null
      || group.isFull() || itemsSize == MAX_ITEMS)
    return false;
  // find an index of the group to be inserted to
  int groupIndex = 0;
  for (; groupIndex < groupsSize && groups[groupIndex] != group; groupIndex++)
    ;
  // if this group was not found in the list, return false
  if (groupIndex == groupsSize)
    return false;

  // find a corresponded place for the item due to order
  int index = 0;
  for (; index < itemsSize
      && (items[index].getDueDate().before(item.getDueDate()) || (items[index]
          .getDueDate().compareTo(item.getDueDate()) == 0 && (items[index]
          .getStatus() < item.getStatus() || (items[index].getStatus() == item
          .getStatus() && items[index].getDescription().compareToIgnoreCase(
          item.getDescription()) <= 0)))); index++)
    // if this item already in the list, return false
    if (items[index] == item)
      return false;
```

```
    // insert the item into the group. Return false if it was not inserted
    if (!group.insert(item))
      return false;

    // shift all items from the right side by one and insert our input item
    for (int i = itemsSize++; i > index; i--)
      items[i] = items[i - 1];
    items[index] = item;
    return true;
  }

  /** Prints the items of the list according to the order of items. */
  public void printAll() {
    if (itemsSize == 0) {
      System.out.println(NO_ITEMS_ERRORS_STRING);
      return;
    }
    for (int i = 0; i < itemsSize; i++)
      items[i].print();
  }

  /** Prints the items of the list by group according to the order of groups and
   * items therein. */
  public void printInGroups() {
    if (groupsSize == 0) {
      System.out.println(NO_GROUPS_ERRORS_STRING);
      return;
    }
    for (int i = 0; i < groupsSize; i++) {
      System.out.println();
      groups[i].printAll();
    }
  }

  /** Prints the list identification number and items of the list by group
   * according to the order of groups and items therein. */
  public void print() {
    System.out.printf(PRINT_FORMAT_STRING, id);
    printInGroups();
  }
}
```

- Group.java

```
package uebung7.question2;

/** The class Group holds a list of its items and provide methods, which allow
 * inserting, sorting, and printing of its items.
 *
 * @author Andrii Dzhyrma */
public class Group {

  // Private Members ////////////////////////////////////////////////////////

  // Constants ---------------------------------------------------------------
  private static final int MAX_ITEMS = 4;
  private static final String PRINT_FORMAT_STRING = "#%d, %s, %d%n--------%n";
  private static final String GROUP_STRING = "Group #";

  // Static Fields -----------------------------------------------------------
  private static int currentId = 1;

  // Fields ------------------------------------------------------------------
  private final Item[] items = new Item[MAX_ITEMS];
  private int itemsSize = 0;
  private int sortOrder = 0;
  private int id;
  private int priority;
  private String name;

  // Public Members /////////////////////////////////////////////////////////

  // Constructors ------------------------------------------------------------
  /** This constructor initializes and generates id for the object.
   *
   * @param name - the name of the group
   * @param priority - the priority of the group (affects on order in the list) */
  public Group(String name, int priority) {
```

```
  id = currentId++;
  this.name = (name != null) ? name : GROUP_STRING + id;
  this.priority = (priority > 0) ? priority : 0;
}

// Methods -----------------------------------------------------------------
/** Returns the name value of the current group.
 *
 * @return the name value */
public String getName() {
  return name;
}

/** Returns the priority value of the current group.
 *
 * @return the priority value */
public int getPriority() {
  return priority;
}

/** Inserts an item into the group storing the item in the group's items-array.
 * The items within a group will be resorted according to the last applied
 * order. <br>
 * <b>Note: it is intended that this method is called by List.insertItem()
 * only! Otherwise some inconsistencies may occur.</b>
 *
 * @param item - the item to be inserted
 * @return In case no insertion can be made false should be returned,
 *         otherwise true. */
public boolean insert(Item item) {
  // if the current array of items is full, or reference to the item is equal
  // to null, or the item is already belongs to another group, return false
  if (itemsSize == MAX_ITEMS || item == null || item.getBelongTo() != null)
    return false;
  // if the input item is already in the list, return false
  for (int i = 0; i < itemsSize; i++)
    if (items[i] == item)
      return false;
  // otherwise add this item into the array and call setBelongTo() method
  items[itemsSize++] = item;
  item.setBelongTo(this);
  // sort the array again
  switch (sortOrder) {
  case 0:
    sortByStatus();
    break;
  case 1:
    sortByDueDate();
  default:
    break;
  }
  return true;
}

/** Checks whether the list of items is empty
 *
 * @return */
public boolean isEmpty() {
  return itemsSize == 0;
}

/** Checks whether the list of items is full
 *
 * @return */
public boolean isFull() {
  return itemsSize == MAX_ITEMS;
}

/** Prints the items of the group according to the order of items. */
public void printAll() {
  System.out.printf(PRINT_FORMAT_STRING, id, name, priority);
  for (int i = 0; i < itemsSize; i++)
    items[i].print();
}

/** Resorts a group's items to order by 1) due-date, 2) status, and 3)
 * description. */
public void sortByDueDate() {
```

```
    // store the order number in the variable
    sortOrder = 1;
    // using algorithm of a minimal element sort the array
    for (int i = 0; i < itemsSize - 1; i++) {
      int minIndex = i;
      for (int j = i + 1; j < itemsSize; j++) {
        int dueDateComparsion = items[j].getDueDate().compareTo(
            items[minIndex].getDueDate());
        if (dueDateComparsion < 0)
          minIndex = j;
        else if (dueDateComparsion == 0
            && (items[j].getStatus() < items[minIndex].getStatus() || (items[j]
                .getStatus() == items[minIndex].getStatus() && items[j]
                .getDescription().compareToIgnoreCase(
                    items[minIndex].getDescription()) < 0)))
          minIndex = j;
      }
      if (minIndex != i) {
        Item temp = items[minIndex];
        items[minIndex] = items[i];
        items[i] = temp;
      }
    }
  }

  /** Resorts a group's items to order by 1) status, 2) due date and 3)
   * description. */
  public void sortByStatus() {
    // store the order number in the variable
    sortOrder = 0;
    // using algorithm of a minimal element sort the array
    for (int i = 0; i < itemsSize - 1; i++) {
      int minIndex = i;
      for (int j = i + 1; j < itemsSize; j++) {
        if (items[j].getStatus() < items[minIndex].getStatus())
          minIndex = j;
        else if (items[j].getStatus() == items[minIndex].getStatus()) {
          int dueDateComparsion = items[j].getDueDate().compareTo(
              items[minIndex].getDueDate());
          if (dueDateComparsion < 0
              || (dueDateComparsion == 0 && items[j].getDescription()
                  .compareToIgnoreCase(items[minIndex].getDescription()) < 0))
            minIndex = j;
        }
      }
      if (minIndex != i) {
        Item temp = items[minIndex];
        items[minIndex] = items[i];
        items[i] = temp;
      }
    }
  }
}
```

- Item.java

```
package uebung7.question2;

import java.util.Calendar;

/** The class Item allows for items to consist a textual description, a due date
 * and a status. Can be added only into an one group.
 *
 * @author Andrii Dzhyrma */
public class Item {

  // Private Members /////////////////////////////////////////////////////////

  // Constants ----------------------------------------------------------------
  private static final String NO_GROUP_BELONG_TO_MESSAGE = "<no group>";
  private static final String PRINT_FORMAT_STRING = "#%d, %s, %s, %d.%02d.%02d, %s%n";
  private static final String[] STATUS_ARRAY = { "open", "started", "finished" };

  // Static Fields ------------------------------------------------------------
  private static int currentId = 1;

  // Fields -------------------------------------------------------------------
  private Group belongTo;
```

```
  private String description;
  private Calendar dueDate;
  private int id;
  private int status;

  // Public Members ////////////////////////////////////////////////////////

  // Constructors -------------------------------------------------------------
  /** This constructor initializes and generates id for the object.
   *
   * @param description - the description of the item
   * @param dueDate - the date of the item event */
  public Item(String description, Calendar dueDate) {
    id = currentId++;
    this.description = (description != null) ? description : "";
    // a status of the item
    this.status = 0;
    // if a dueDate value is null, generate current the current time instead
    if (dueDate == null)
      this.dueDate = Calendar.getInstance();
    else
      this.dueDate = (Calendar) dueDate.clone();
  }

  /** Finish the current event */
  public void finish() {
    status = 2;
  }

  /** Returns the group to what this item belongs to
   *
   * @return - the parent group */
  public Group getBelongTo() {
    return belongTo;
  }

  /** Returns the description of the current item
   *
   * @return - the description value */
  public String getDescription() {
    return description;
  }

  /** Returns the date of the item event to be expired
   *
   * @return - the dueDate value */
  public Calendar getDueDate() {
    return dueDate;
  }

  /** Returns the status of the item interpreted as an integer value
   *
   * @return - the status value */
  public int getStatus() {
    return status;
  }

  /** Prints the item and the information about the group it belongs to. */
  public void print() {
    System.out.printf(PRINT_FORMAT_STRING, id, description,
        (belongTo == null) ? NO_GROUP_BELONG_TO_MESSAGE : belongTo.getName(),
        dueDate.get(Calendar.YEAR), dueDate.get(Calendar.MONTH) + 1,
        dueDate.get(Calendar.DAY_OF_MONTH), STATUS_ARRAY[status]);
  }

  /** Sets the parent group of the item
   *
   * @param belongTo - the group value */
  public void setBelongTo(Group belongTo) {
    this.belongTo = belongTo;
  }

  /** Starts the item event */
  public void start() {
    status = 1;
  }
}
```

- Main.java

```
package uebung7.question2;

import java.util.GregorianCalendar;

public class Main {
  private static final String GROUP_INSERTED_SUCCESFULLY_FORMAT_STRING = "Group \"%s\" has been inserted
succesfully!%n";
  private static final String GROUP_INSERTION_FAILED_FORMAT_STRING = "Group \"%s\" insertion failed!%n";
  private static final String ITEM_INSERTED_SUCCESFULLY_FORMAT_STRING = "Item \"%s\" has been inserted
succesfully!%n";
  private static final String ITEM_INSERTION_FAILED_FORMAT_STRING = "Item \"%s\" insertion failed!%n";

  private static void insertGroup(Group group, List list) {
    if (list.insertGroup(group))
      System.out.printf(GROUP_INSERTED_SUCCESFULLY_FORMAT_STRING,
          (group == null) ? "with null value" : group.getName());
    else
      System.out.printf(GROUP_INSERTION_FAILED_FORMAT_STRING,
          (group == null) ? "with null value" : group.getName());
  }

  private static void insertItem(Item item, Group group, List list) {
    if (list.insertItem(item, group))
      System.out.printf(ITEM_INSERTED_SUCCESFULLY_FORMAT_STRING,
          (item == null) ? "with null value" : item.getDescription());
    else
      System.out.printf(ITEM_INSERTION_FAILED_FORMAT_STRING,
          (item == null) ? "with null value" : item.getDescription());
  }

  /** @param args - no arguments will evaluate */
  public static void main(String[] args) {
    List list1 = new List();
    System.out.println("Printing an empty list (printAll):");
    list1.printAll();
    System.out.println("Printing an empty list (printInGroups):");
    list1.printInGroups();
    Group work = new Group("work", 1);
    System.out.println("Printing an empty group:");
    work.printAll();
    Group study = new Group("study", 2);
    Group shopping = new Group("shopping", 3);
    Item item1 = new Item("prepare exrcise", new GregorianCalendar(2012, 10, 24));
    System.out.println("Printing an item:");
    item1.print();
    System.out.println();
    item1.finish();
    Item item2 = new Item("correct test", new GregorianCalendar(2012, 11, 01));
    Item item3 = new Item("read book JAVA 7", new GregorianCalendar(2012, 11, 24));
    Item item4 = new Item("buy milk", new GregorianCalendar(2012, 11, 24));
    Item item5 = new Item("fetch book", new GregorianCalendar(2012, 11, 24));
    item5.finish();
    insertGroup(work, list1);
    insertGroup(study, list1);
    insertGroup(shopping, list1);
    insertItem(item1, work, list1);
    insertItem(item2, work, list1);
    insertItem(item3, study, list1);
    insertItem(item4, shopping, list1);
    insertItem(item5, shopping, list1);
    List list2 = new List();
    Group group1 = new Group(null, -1);
    Group group2 = new Group("Group", 2);
    Group group3 = new Group("Group", 2);
    Group group4 = new Group("Group4", 3);
    Group group5 = new Group("Group5", 3);
    Item item6 = new Item(null, null);
    Item item7 = new Item("Item", null);
    Item item8 = new Item("Item", new GregorianCalendar(2012, 11, 24));
    Item item9 = new Item("Item", new GregorianCalendar(2012, 11, 24));
    Item item10 = new Item("Item10", new GregorianCalendar(2012, 12, 24));
    Item item11 = new Item("Item11", new GregorianCalendar(2013, 12, 24));
    Item item12 = new Item("Item12", new GregorianCalendar(2013, 12, 23));
    item8.start();
    item9.start();
```

```
        item9.finish();
        insertGroup(null, list2);
        insertGroup(group1, list2);
        insertGroup(group2, list2);
        insertGroup(group3, list2);
        insertGroup(group3, list2);
        insertGroup(work, list2);
        insertGroup(group4, list2);
        insertGroup(group5, list2);
        insertItem(null, null, list2);
        insertItem(null, group1, list2);
        insertItem(item6, null, list2);
        insertItem(item1, work, list2);
        insertItem(item6, work, list2);
        insertItem(item1, group1, list2);
        insertItem(item6, group1, list2);
        insertItem(item7, group1, list2);
        insertItem(item8, group1, list2);
        insertItem(item9, group1, list2);
        insertItem(item10, group1, list2);
        insertItem(item10, group2, list2);
        insertItem(item11, group2, list2);
        insertItem(item12, group2, list2);

        System.out.println("\nPrinting the list1 (printAll):");
        list1.printAll();
        System.out.println("\nPrinting the list1 (printInGroups):");
        list1.printInGroups();
        System.out.println("\nPrinting the list2 (printAll):");
        list2.printAll();
        System.out.println("\nPrinting the list2 (printInGroups):");
        list2.printInGroups();

        group1.sortByDueDate();
        System.out.println("\nPrinting the group sorted by dueDate:");
        group1.printAll();
        group1.sortByStatus();
        System.out.println("\nPrinting the group sorted by status:");
        group1.printAll();
    }
}
```

c) Test plan

#	Description	Expected output
1	Test printing methods of the class list and the group without elements	**Get an error about the amount of elements**
2	Modulate the example from the requirements	**Get the same result as written in the requirements**
3	Test constructors of the group and the item for the null and incorrect input values	**Create objects with adequate values**
4	Test the “insert” methods for the null, already existing items and space limitations	**Return false in incorrect cases**
5	Test the sorting methods in the class group	**A correspondent reaction as described in the requirements**

d) The output of the program

Output	Comments
Printing an empty list (printAll): There are no items in this list.	- list1.printAll();
Printing an empty list (printInGroups): There are no groups in this list.	- list1.printInGroups();
Printing an empty group: #1, work, 1 --------	- work.printAll();
Printing an item: #1, prepare exrcise, <no group>, 2012.11.24, open	- item1.print()
Group "work" has been inserted succesfully!	- insertGroup(work, list1);
Group "study" has been inserted succesfully!	- insertGroup(study, list1);
Group "shopping" has been inserted succesfully!	- insertGroup(shopping, list1);
Item "prepare exrcise" has been inserted succesfully!	- insertItem(item1, work, list1);
Item "correct test" has been inserted succesfully!	- insertItem(item2, work, list1);
Item "read book JAVA 7" has been inserted succesfully!	- insertItem(item3, study, list1);
Item "buy milk" has been inserted succesfully!	- insertItem(item4, shopping, list1);
Item "fetch book" has been inserted succesfully!	- insertItem(item5, shopping, list1);
Group "with null value" insertion failed!	- insertGroup(null, list2); //null
Group "Group #4" has been inserted succesfully!	- insertGroup(group1, list2);
Group "Group" has been inserted succesfully!	- insertGroup(group2, list2);
Group "Group" has been inserted succesfully!	- insertGroup(group3, list2);
Group "Group" insertion failed!	- insertGroup(group3, list2); //exists
Group "work" insertion failed!	- insertGroup(work, list2); //in the list1
Group "Group4" has been inserted succesfully!	- insertGroup(group4, list2);
Group "Group5" insertion failed!	- insertGroup(group5, list2); //space limit
Item "with null value" insertion failed!	- insertItem(null, null, list2); //null
Item "with null value" insertion failed!	- insertItem(null, group1, list2); //null
Item "" insertion failed!	- insertItem(item6, null, list2); //null
Item "prepare exrcise" insertion failed!	- insertItem(item1, work, list2); //in the list1
Item "" insertion failed!	- insertItem(item6, work, list2); //in the list1
Item "prepare exrcise" insertion failed!	- insertItem(item1, group1,list2);//in the list1
Item "" has been inserted succesfully!	- insertItem(item6, group1, list2);
Item "Item" has been inserted succesfully!	- insertItem(item7, group1, list2);
Item "Item" has been inserted succesfully!	- insertItem(item8, group1, list2);
Item "Item" has been inserted succesfully!	- insertItem(item9, group1, list2);
Item "Item10" insertion failed!	- insertItem(item10, group1,list2);//space limit
Item "Item10" has been inserted succesfully!	- insertItem(item10, group2, list2);
Item "Item11" has been inserted succesfully!	- insertItem(item11, group2, list2);
Item "Item12" insertion failed!	- insertItem(item12, group2,list2);//space limit
Printing the list1 (printAll): #1, prepare exrcise, work, 2012.11.24, finished #2, correct test, work, 2012.12.01, open #4, buy milk, shopping, 2012.12.24, open #3, read book JAVA 7, study, 2012.12.24, open #5, fetch book, shopping, 2012.12.24, finished	- list1.printAll();
Printing the list1 (printInGroups): #1, work, 1 -------- #2, correct test, work, 2012.12.01, open #1, prepare exrcise, work, 2012.11.24, finished #2, study, 2 -------- #3, read book JAVA 7, study, 2012.12.24, open #3, shopping, 3 -------- #4, buy milk, shopping, 2012.12.24, open #5, fetch book, shopping, 2012.12.24, finished	- list1.printInGroups();
Printing the list2 (printAll): #8, Item, Group #4, 2012.12.24, started #9, Item, Group #4, 2012.12.24, finished #6, , Group #4, 2013.01.09, open #7, Item, Group #4, 2013.01.09, open #10, Item10, Group, 2013.01.24, open #11, Item11, Group, 2014.01.24, open	- list2.printAll();
Printing the list2 (printInGroups):	- list2.printInGroups();

#4, Group #4, 0
-------- 	
#6, , Group #4, 2013.01.09, open
#7, Item, Group #4, 2013.01.09, open
#8, Item, Group #4, 2012.12.24, started
#9, Item, Group #4, 2012.12.24, finished
 	
#5, Group, 2
-------- 	
#10, Item10, Group, 2013.01.24, open
#11, Item11, Group, 2014.01.24, open
 	
#6, Group, 2
-------- 	
 	
#7, Group4, 3
-------- 	
 	
Printing the group sorted by dueDate:
#4, Group #4, 0
-------- 	
#8, Item, Group #4, 2012.12.24, started
#9, Item, Group #4, 2012.12.24, finished
#6, , Group #4, 2013.01.09, open
#7, Item, Group #4, 2013.01.09, open
 	
Printing the group sorted by status:
#4, Group #4, 0
-------- 	
#6, , Group #4, 2013.01.09, open
#7, Item, Group #4, 2013.01.09, open
#8, Item, Group #4, 2012.12.24, started
#9, Item, Group #4, 2012.12.24, finished
	- group1.sortByDueDate();
	  group1.printAll();
	
	- group1.sortByStatus();
	  group1.printAll();