Question 1: 2D-Diagrams

a) The idea of my solution.

First we initialize important constants (maximal values for the input point coordinate, step for the grid, left margin and strings). Then we print out description of the program and X and Y coordinate of the point. In while loop we check are the values out of bounds and if yes, continuously ask user to input correct value. Then we print out Y-axis with checking for the point coordinate X: if X is equal to 0, we should print point on the axis instead of the special symbol. Each grid step we also print correspondent value before the special symbol. If we reached Y coordinate of the point, we just print out our star symbol with equal to X value offset. Then we print out X-axis with the same checking for Y value, if it is equal 0. After that we print out correspondent grid values of the X-axis below it.

b) Source code

```
package uebung5.question1;

import io.Input;

/**
 * @author Andrii Dzhyrma
 */
public class Diagram2D {

 // All the important constants
 private static final int MAX_X = 20;
 private static final int MAX_Y = 15;
 private static final int GRID_STEP = 5;
 private static final int LEFT_MARGIN = 2;
 private static final char POINT_CHAR = '*';

 // Constant string with description of the program
 private static final String PROGRAM_DESCRIPTION_STRING = "This program will place a
point on a plot.";
 // Constant requesting for coordinate string
 private static final String COORDINATE_REQUEST_FORMAT_STRING = "Enter the %c-
coordinate:%n";
 // Error message for input coordinate being out of range
 private static final String INVALID_COORDINATE_RANGE_ERROR_STRING = "Coordinate
should be in range [%d, %d]!%n";
 // Coordinate value format string
 private static final String COORDINATE_VALUE_FORMAT_STRING = "%%%dd";
 // Pseudo drawing format string
 private static final String DRAWING_FORMAT_STRING = "%%%dc";

 /**
  * @param args
  *            - no arguments will evaluate
  */
 public static void main(String[] args) {
  // Print out the description of the program
  System.out.println(PROGRAM_DESCRIPTION_STRING);

  // Read coordinates x and y
  int x, y;
  System.out.printf(COORDINATE_REQUEST_FORMAT_STRING, 'x');
  x = Input.readInt();
  while (x < 0 || x > MAX_X) {
   System.out.printf(INVALID_COORDINATE_RANGE_ERROR_STRING, 0, MAX_X);
```

```java
    System.out.printf(COORDINATE_REQUEST_FORMAT_STRING, 'x');
    x = Input.readInt();
   }
   System.out.printf(COORDINATE_REQUEST_FORMAT_STRING, 'y');
   y = Input.readInt();
   while (y < 0 || y > MAX_Y) {
    System.out.printf(INVALID_COORDINATE_RANGE_ERROR_STRING, 0, MAX_Y);
    System.out.printf(COORDINATE_REQUEST_FORMAT_STRING, 'y');
    y = Input.readInt();
   }

   // Print out the Y-axis
   System.out.println();
   System.out.printf(String.format(DRAWING_FORMAT_STRING, LEFT_MARGIN + 1), '^');
   System.out.println();
   for (int i = MAX_Y; i > 0; i--) {
    // Print out Y numbers for the grid
    if (i % GRID_STEP == 0)
     System.out.printf(String.format(COORDINATE_VALUE_FORMAT_STRING, LEFT_MARGIN), i);
    else
     System.out.printf(String.format(DRAWING_FORMAT_STRING, LEFT_MARGIN), ' ');
    // If our point is on axis, print out star symbol on it
    if (y == i && x == 0)
     System.out.println(POINT_CHAR);
    // In other case just draw an axis and point if it's y coordinate equal to i
    else {
     if (i % GRID_STEP == 0)
      System.out.print('+');
     else
      System.out.print('|');
     if (y == i)
      System.out.printf(String.format(DRAWING_FORMAT_STRING, x), POINT_CHAR);
     System.out.println();
    }
   }
   // Print out the X-axis
   System.out.printf(String.format(DRAWING_FORMAT_STRING, LEFT_MARGIN), ' ');
   for (int i = 0; i <= MAX_X; i++) {
    // If our point is on axis X, print out start symbol on it
    if (y == 0 && x == i)
     System.out.print(POINT_CHAR);
    else if (i % GRID_STEP == 0)
     System.out.print('+');
    else
     System.out.print('-');
   }
   System.out.println('>');
   // Print out X numbers for the grid
   System.out.printf(String.format(DRAWING_FORMAT_STRING, LEFT_MARGIN), ' ');
   for (int i = 0; i <= MAX_X; i += GRID_STEP)
    System.out.printf(String.format(COORDINATE_VALUE_FORMAT_STRING, -GRID_STEP), i);
  }
 }
```

c) Test plan

#	Aim	Input	Expected output
1	Common case	5, 5	**Normally printed point as in the example**
2	Common case	20, 15	**Normally printed point on the correspondent coordinates**
3	Common case	10, 0	**Printed on the X-axis point**
4	Common case	0, 14	**Printed on the Y-axis point**
5	Common case	0, 0	**Printed on the corner point**
6	Coordinates out of bounds	-1, 16	**Printed out the error and ask again**
7	Float type values	0.5, 1.5	**Printed out the error and ask again**
8	String type values	abcd, qwer	**Printed out the error and ask again**
9	Value bigger than maximum integer value	2147483648	**Printed out the error and ask again**

d) The output of the program

#1

```
This program will place a
point on a plot.
Enter the x-coordinate:
5
Enter the y-coordinate:
5

  ^
15+
  |
  |
  |
  |
10+
  |
  |
  |
  |
 5+    *
  |
  |
  |
  |
  +----+----+----+----+>
  0    5    10   15   20
```

#2

```
This program will place a
point on a plot.
Enter the x-coordinate:
20
Enter the y-coordinate:
15

  ^
15+                   *
  |
  |
  |
  |
10+
  |
  |
  |
  |
 5+
  |
  |
  |
  |
  +----+----+----+----+>
  0    5    10   15   20
```

#3

```
This program will place a
point on a plot.
Enter the x-coordinate:
10
Enter the y-coordinate:
0

  ^
15+
  |
  |
  |
  |
10+
  |
  |
  |
  |
 5+
  |
  |
  |
  |
  +----+----*----+----+>
  0    5    10   15   20
```

#4

```
This program will place a
point on a plot.
Enter the x-coordinate:
```

#5

```
This program will place a
point on a plot.
Enter the x-coordinate:
```

#6

```
This program will place a
point on a plot.
Enter the x-coordinate:
```

0
Enter the y-coordinate:
14
 	
  ^
15+
  *
  |
  |
  |
10+
  |
  |
  |
  |
  5+
  |
  |
  |
  |
  +----+----+----+----+>
  0 5 10 15 20
	0
	Enter the y-coordinate:
	0
	
	  ^
	15+
	  |
	  |
	  |
	  |
	10+
	  |
	  |
	  |
	  |
	  5+
	  |
	  |
	  |
	  |
	  *----+----+----+----+>
	  0 5 10 15 20
		-1
		Coordinate should be in range [0, 20]!
		Enter the x-coordinate:
		1
		Enter the y-coordinate:
		16
		Coordinate should be in range [0, 15]!
		Enter the y-coordinate:

#7 This program will place a point on a plot. Enter the x-coordinate: 0.5 Not a valid int, please try again: 1.5 Not a valid int, please try again:	**#8** This program will place a point on a plot. Enter the x-coordinate: abcd Not a valid int, please try again: qwer Not a valid int, please try again:	**#9** This program will place a point on a plot. Enter the x-coordinate: 2147483648 Not a valid int, please try again:

Question 2: Grid of Emoticons

a) The idea of my solution.

First we initialize all important constants (given emoticons, number of maximum iterations, number of emoticons to choose and strings for output to optimize memory usage). Then we initialize variables (random generator, jagged arrays, array for choosing emoticons, and each array for the second dimension in the jagged array). Then we read chosen by user emoticons and if some of them is out of range, print error and ask to input one more time. To randomly place all emoticons into the jagged array, we create Boolean variable ‘fail’ and if at the end of randomization it will be false, then every emoticon was placed correctly. For the case when we will reach maximum amount of iteration and there will be no available position for all chosen emoticons, we create variable maxPlacedEmoticons to know, the best result of positioning. At the iteration of placing, first we assign to ‘fail’ variable value ‘false’ and fill the jagged array with dashes. In the next loop for each chosen emoticon we calculate all possible positions to place it in the grid according only to positioning rules without overlapping. Then randomly we choose one of the position if it exists and check if on that position is already another written emoticon or not. If not, we write current emoticon on that position. After this loop we check if this number of emoticons we wrote to the grid bigger than previous or ‘fail’ variable is still has value ‘false’, we copying jaggedWorkingArray to jaggedResultArray. At the end we print out our result jagged array (grid) and if not all emoticons were placed, also an error message about that.

b) Source code

```java
package uebung5.question2;

import io.Input;
import java.util.Arrays;
import java.util.Random;

/**
 * @author Andrii Dzhyrma
 */
public class GridOfEmoticons {

 // All the important constants
 static final char[][] EMOTICONS = { { 'n', '_', 'n' }, { '$', 'v', '$' },
   { '8', '(', '>', '_', '<', ')', '8' },
   { 'W', '(', '^', 'o', '^', ')', 'W' }, { '(', '=', '_', '=', ')' },
   { '(', '/', '_', '\\', ')' }, { '>', '^', '.', '.', '^', '<' },
   { '(', '~', '-', '^', ')' }, { '(', '*', '-', '*', ')' },
   { '<', '*', ')', ')', ')', '-', '{' } };
 static final int NUMBER_OF_MAX_ITERATIONS = 100000;
 static final int NUMBER_OF_CHOSEN_EMOTICONS = 5;
 static final String GRID_DESCRIPTION_STRING = "This is the generated grid:";
 static final String NUMBER_OF_EMOTICON_OUT_OF_RANGE_ERROR_STRING = "Chosen emoticon
does not exist. Chose one from the list above:";
 static final String PLACE_EMOTICONS_ERROR_STRING = "Not all emoticons could be
placed!";
 static final String SELECT_EMOTICONS_DESCRIPTION_STRING_FORMAT = "Please select %d
emoticons from the following list:%n";
 static final String SELECT_EMOTICONS_STRING_FORMAT = "Select emoticon # %d%n";
 static final String NUMERATION_STRING_FORMAT = "%d: ";

 /**
  * @param args
  *            - no arguments will evaluate
  */
 public static void main(String[] args) {
  // Initialize the variables
  Random rand = new Random();
  char[][] jaggedWorkingArray = new char[10][];
  char[][] jaggedResultArray = new char[10][];
  int[] chosenEmoticons = new int[NUMBER_OF_CHOSEN_EMOTICONS];
  // Initialize the jagged array grid
  for (int i = 0; i < jaggedWorkingArray.length; i++)
   jaggedWorkingArray[i] = new char[rand.nextInt(9) + 2];

  // Print out all the emoticons
  System.out.printf(SELECT_EMOTICONS_DESCRIPTION_STRING_FORMAT,
    NUMBER_OF_CHOSEN_EMOTICONS);
  for (int i = 0; i < EMOTICONS.length; i++) {
   System.out.printf(NUMERATION_STRING_FORMAT, i + 1);
   for (int j = 0; j < EMOTICONS[i].length; j++)
    System.out.print(EMOTICONS[i][j]);
   System.out.println();
  }
  // Read 5 emoticons chosen by user
  for (int i = 0; i < NUMBER_OF_CHOSEN_EMOTICONS; i++) {
   System.out.printf(SELECT_EMOTICONS_STRING_FORMAT, i + 1);
   chosenEmoticons[i] = Input.readInt() - 1;
   while (chosenEmoticons[i] < 0
     || chosenEmoticons[i] >= EMOTICONS.length) {
    System.out
```

```
    .println(NUMBER_OF_EMOTICON_OUT_OF_RANGE_ERROR_STRING);
  chosenEmoticons[i] = Input.readInt() - 1;
 }
}
System.out.println();

// Initialize 'fail' variable to know are all emoticons placed in the
// grid
boolean fail = true;
// Initialize 'maxPlacedEmoticons' variable to save better result at the
// end
int maxPlacedEmoticons = 0;
// This loop is for trying to put emoticons randomly to the grid
for (int i = 0; fail && i < NUMBER_OF_MAX_ITERATIONS; i++) {
 // Assume that this time we will put emoticons correctly
 fail = false;
 // Fill the grid array with dashes
 for (int j = 0; j < jaggedWorkingArray.length; j++)
  Arrays.fill(jaggedWorkingArray[j], '-');

 int j;
 // For each chosen emoticon try to find random position
 for (j = 0; !fail && j < NUMBER_OF_CHOSEN_EMOTICONS; j++) {
  // Calculate possible places for emoticon corresponding to the
  // size of each row in the grid
  int possibleCoordinates = 0;
  for (int k = 0; k < jaggedWorkingArray.length; k++) {
   if (EMOTICONS[chosenEmoticons[j]].length <= jaggedWorkingArray[k].length)
    possibleCoordinates += jaggedWorkingArray[k].length
      - EMOTICONS[chosenEmoticons[j]].length + 1;
  }
  // If we did not find any place for emoticon, we should start
  // again
  if (possibleCoordinates == 0)
   fail = true;
  else {
   // Get random available position for the emoticon
   int column = rand.nextInt(possibleCoordinates);
   int row = 0;
   // Calculation of the row and column where should we put
   // first symbol of the emoticon
   for (row = 0; column > jaggedWorkingArray[row].length
     - EMOTICONS[chosenEmoticons[j]].length
     && row < jaggedWorkingArray.length; row++)
    if (EMOTICONS[chosenEmoticons[j]].length <= jaggedWorkingArray[row].length)
     column -= jaggedWorkingArray[row].length
       - EMOTICONS[chosenEmoticons[j]].length + 1;
   // Check if there is already another emoticon written
   for (int k = 0; !fail
     && k < EMOTICONS[chosenEmoticons[j]].length; k++)
    if (jaggedWorkingArray[row][column + k] != '-')
     fail = true;
   // Write current emoticon to the grid
   for (int k = 0; !fail
     && k < EMOTICONS[chosenEmoticons[j]].length; k++)
    jaggedWorkingArray[row][column + k] = EMOTICONS[chosenEmoticons[j]][k];
  }
 }
 // If the result is better then previous, copying jaggedWorkingArray
 // to the jaggedResultArray
 if (j > maxPlacedEmoticons || !fail) {
  maxPlacedEmoticons = j;
```

```
    for (int k = 0; k < jaggedWorkingArray.length; k++)
     jaggedResultArray[k] = Arrays.copyOf(jaggedWorkingArray[k],
       jaggedWorkingArray[k].length);
   }
  }

  // Print out the result grid
  System.out.println(GRID_DESCRIPTION_STRING);
  for (int i = 0; i < jaggedResultArray.length; i++) {
   for (int j = 0; j < jaggedResultArray[i].length; j++)
    System.out.print(jaggedResultArray[i][j]);
   System.out.println();
  }
  System.out.println();
  // Print out an error if not all the emoticons were displayed
  if (fail)
   System.out.println(PLACE_EMOTICONS_ERROR_STRING);
 }
}
```

c) Test plan

#	Aim	Input	Expected output
1	Common case	1, 2, 3, 4, 5	**Normally printed grid possibly with all emoticons on it**
2	Common case	1, 1, 1, 1, 1	**Normally printed grid possibly with all emoticons on it**
3	Common case	10, 10, 10, 10, 10	**Normally printed grid possibly with all emoticons on it**
4	Chosen emoticon does not exists	<1 or >10	**Printed out the error and ask again**
5	Float type values	0.5	**Printed out the error and ask again**
6	String type values	abcd	**Printed out the error and ask again**
7	Value bigger than maximum integer value	2147483648	**Printed out the error and ask again**

d) The output of the program

#1	#2	#3
Please select 5 emoticons from the following list: 1: n_n 2: v 3: 8(>_<)8 4: W(^o^)W 5: (=_=) 6: (/_\) 7: >^..^< 8: (~-^) 9: (*-*) 10: <*)))-{ Select emoticon # 1 1 Select emoticon # 2	Please select 5 emoticons from the following list: 1: n_n 2: v 3: 8(>_<)8 4: W(^o^)W 5: (=_=) 6: (/_\) 7: >^..^< 8: (~-^) 9: (*-*) 10: <*)))-{ Select emoticon # 1 1 Select emoticon # 2	Please select 5 emoticons from the following list: 1: n_n 2: v 3: 8(>_<)8 4: W(^o^)W 5: (=_=) 6: (/_\) 7: >^..^< 8: (~-^) 9: (*-*) 10: <*)))-{ Select emoticon # 1 10 Select emoticon # 2

2 Select emoticon # 3 3 Select emoticon # 4 4 Select emoticon # 5 5 This is the generated grid: --------- -v------ ---- ---(=_=)-- -- ---n_n -------- -8(>_<)8- ------ W(^o^)W--	1 Select emoticon # 3 1 Select emoticon # 4 1 Select emoticon # 5 1 This is the generated grid: -- n_n-- ------ ----- -n_n-n_n-- ---n_n -n_n -------- ----- -----	10 Select emoticon # 3 10 Select emoticon # 4 10 Select emoticon # 5 10 This is the generated grid: ---- --<*)))-{- --<*)))-{- -- <*)))-{--- -- -- ----- ---- ------ Not all emoticons could be placed!
#4 Please select 5 emoticons from the following list: 1: n_n 2: v 3: 8(>_<)8 4: W(^o^)W 5: (=_=) 6: (/_\) 7: >^..^< 8: (~-^) 9: (*-*) 10: <*)))-{ Select emoticon # 1 0 Chosen emoticon does not exist. Chose one from the list above: 11 Chosen emoticon does not exist. Chose one from the list above:	**#5** Please select 5 emoticons from the following list: 1: n_n 2: v 3: 8(>_<)8 4: W(^o^)W 5: (=_=) 6: (/_\) 7: >^..^< 8: (~-^) 9: (*-*) 10: <*)))-{ Select emoticon # 1 0.5 Not a valid int, please try again:	**#6** Please select 5 emoticons from the following list: 1: n_n 2: v 3: 8(>_<)8 4: W(^o^)W 5: (=_=) 6: (/_\) 7: >^..^< 8: (~-^) 9: (*-*) 10: <*)))-{ Select emoticon # 1 abcd Not a valid int, please try again:
#7 Please select 5 emoticons from the following list: 1: n_n 2: v 3: 8(>_<)8 4: W(^o^)W 5: (=_=) 6: (/_\) 7: >^..^< 8: (~-^) 9: (*-*) 10: <*)))-{ Select emoticon # 1 2147483648 Not a valid int, please try again:		