Question 1: Find Your Way

a) The idea of my solution

In the class Position to prevent calculations all the time when we call “blocked” and “outOfBounds” we create two private Boolean fields “isBlocked” and “isOudOfBounds” which we calculate in the constructor. The maze we store in this class. Also we implement getters for the coordinate X and the coordinate Y.

In the class Main we create a two dimensional array to store visited cells. We print the maze as shown in the requirements and ask user to input the coordinates for start and destination positions. Then we call a recursive function “findPassage”. This function first checks whether current cell is blocked, destination cell is blocked, does current cell positioning on the left side from the destination cell and is current cell visited. If all conditions are correct we mark current cell as visited with the number “2”, check whether we reached the destination cell already and if not, check recursively with the “findPassage” function exist or not passages from the three neighbor cells (up, right and down cells). If the result is true, than mark the current cell as “1” in the visited array.

If “findPassage” returned true for the given positions, we draw a maze with plus symbols instead of numbers “1” from the array of visited states and otherwise we draw symbols from the maze.

b) Source code

- Main.java

```
package uebung10.question1;

import io.Input;

public class Main {

  // array to store visited cells
  private static int[][] visited;

  /** @param args - no arguments will evaluate */
  public static void main(String[] args) {
    // if the maze is invalid, do nothing
    if (Position.MAZE == null || Position.MAZE.length == 0) {
      System.out.println("The Maze is empty!");
      return;
    }
    // initialize the array for visited states
    visited = new int[Position.MAZE.length][];
    for (int i = 0; i < visited.length; i++)
      visited[i] = new int[Position.MAZE[i].length];
    // print the maze on the screen
    System.out.println("The Maze");
    for (int i = 0; i < Position.MAZE[0].length + 2; i++)
      System.out.print('-');
    System.out.println();
    for (int i = 0; i < Position.MAZE.length; i++) {
      String line = String.copyValueOf(Position.MAZE[i]);
      System.out.println('|' + line + '|');
    }
    for (int i = 0; i < Position.MAZE[Position.MAZE.length - 1].length + 2; i++)
      System.out.print('-');
    // read coordinates for start and destination points
    int x, y;
    System.out.println();
    System.out.print("Find a passage from start [x]: ");
    x = Input.readInt();
    System.out.print("Find a passage from start [y]: ");
    y = Input.readInt();
```

```
    Position start = new Position(x, y);
    System.out.print("To destination [x]: ");
    x = Input.readInt();
    System.out.print("To destination [y]: ");
    y = Input.readInt();
    Position destination = new Position(x, y);

    // if a passage exists do next
    if (findPassage(start, destination)) {
      System.out.println("A passage from [" + start.getX() + "," + start.getY()
          + "] to [" + destination.getX() + "," + destination.getY()
          + "] could be found!");
      // print the maze with the passage
      System.out.println("The passage through the maze is:");
      for (int i = 0; i < Position.MAZE[0].length + 2; i++)
        System.out.print('-');
      System.out.println();
      for (int i = 0; i < Position.MAZE.length; i++) {
        String line = "";
        for (int j = 0; j < Position.MAZE[i].length; j++)
          line += (visited[i][j] == 1) ? '+' : Position.MAZE[i][j];
        System.out.println('|' + line + '|');
      }
      for (int i = 0; i < Position.MAZE[Position.MAZE.length - 1].length + 2; i++)
        System.out.print('-');
    } else {
      // no passage was found
      System.out.println("A passage from [" + start.getX() + "," + start.getY()
          + "] to [" + destination.getX() + "," + destination.getY()
          + "] could not be found!");
    }
  }

  /**
   * Finds a passage from the start position to the destination position.
   * @param start - the start position
   * @param destination - the destination position
   * @return - return true is the passage was found
   */
  public static boolean findPassage(Position start, Position destination) {
    // if blocked, out of bounds, start position is on the right side of the destination or visited,
    // return false
    if (destination.blocked() || destination.outOfBounds() || start.blocked()
        || start.outOfBounds() || start.getX() > destination.getX()
        || visited[start.getY()][start.getX()] > 0)
      return false;
    // mark as visited
    visited[start.getY()][start.getX()] = 2;
    boolean result = start.getX() == destination.getX()
        && start.getY() == destination.getY()
        || findPassage(start.right(), destination)
        || findPassage(start.down(), destination)
        || findPassage(start.up(), destination);
    // mark as correct path
    if (result)
      visited[start.getY()][start.getX()] = 1;
    return result;
  }
}
```

- Position.java

```
package uebung10.question1;

/** The Position class gives possibility to go through the maze and check whether
 * a path is blocked or a position is out of bounds.
 *
 * @author Andrii Dzhyrma */
public class Position {
  /** The maze */
  public static final char[][] MAZE = {
      { '_', '_', '_', '_', '_', '_', '_', '_', '#', '_', '_', '_' },
      { '_', '_', '_', '_', '_', '_', '_', '_', '#', '#', '_', '_' },
      { '_', '_', '_', '_', '_', '_', '_', '_', '#', '_', '_', '_' },
      { '_', '_', '_', '_', '_', '_', '_', '_', '#', '_', '#', '_' },
      { '_', '#', '#', '#', '_', '_', '_', '_', '#', '_', '#', '_' },
      { '_', '#', '_', '#', '_', '_', '#', '_', '_', '_', '#', '_' },
      { '_', '#', '#', '#', '_', '_', '#', '_', '_', '_', '_', '_' },
```

```
      { '_', '_', '_', '_', '_', '_', '_', '_', '#', '#', '#', '_' },
      { '_', '_', '_', '_', '_', '#', '_', '_', '_', '_', '_', '_' } };
  private final boolean isBlocked;
  private final boolean isOutOfBounds;
  private final int x;
  private final int y;

  /** Initializes the position.
   *
   * @param x - the X coordinate
   * @param y - the Y coordinate */
  public Position(int x, int y) {
    this.x = x;
    this.y = y;
    // checks for the blocked cell and is it out of bounds
    isOutOfBounds = y < 0 || y >= MAZE.length || x < 0 || x >= MAZE[y].length;
    isBlocked = !isOutOfBounds && MAZE[y][x] == '#';
  }

  /** @return is current cell blocked. */
  public boolean blocked() {
    return isBlocked;
  }

  /** @return the cell with Y coordinate greater by one. */
  public Position down() {
    return new Position(x, y + 1);
  }

  /** @return the X coordinate. */
  public final int getX() {
    return this.x;
  }

  /** @return the Y coordinate. */
  public final int getY() {
    return this.y;
  }

  /** @return the cell with X coordinate less by one. */
  public Position left() {
    return new Position(x - 1, y);
  }

  /** @return true if current cell is out of bounds. */
  public boolean outOfBounds() {
    return isOutOfBounds;
  }

  /** @return the cell with X coordinate greater by one. */
  public Position right() {
    return new Position(x + 1, y);
  }

  /** @return the cell with Y coordinate less by one. */
  public Position up() {
    return new Position(x, y - 1);
  }
}
```

c) Test plan

#	Description	Expected output
1	Test normal inputs	**Correct passages**
2	The start or the destination point is on the wall	**Passages could not be found**
3	The start or the destination point is out of the border	**Passages could not be found**

4	The start or the destination point is blocked	**Passages could not be found**
5	Invalid input	**Ask for the correct integers**

d) The output of the program

#	Input	Output	#	Input	Output
1	0,0,0,0	The Maze -------------- |_____#___| |_____##__| |_____#___| |_____#_#_| |_###___#_#_| |_#_#__#___#_| |_###__#_____| |_____###_| |____#_____| -------------- Find a passage from start [x]: 0 Find a passage from start [y]: 0 To destination [x]: 0 To destination [y]: 0 A passage from [0,0] to [0,0] could be found! The passage through the maze is: -------------- |+_____#___| |_____##__| |_____#___| |_____#_#_| |_###___#_#_| |_#_#__#___#_| |_###__#_____| |_____###_| |____#_____| --------------	1	0,0,11,8	The Maze -------------- |_____#___| |_____##__| |_____#___| |_____#_#_| |_###___#_#_| |_#_#__#___#_| |_###__#_____| |_____###_| |____#_____| -------------- Find a passage from start [x]: 0 Find a passage from start [y]: 0 To destination [x]: 11 To destination [y]: 8 A passage from [0,0] to [11,8] could be found! The passage through the maze is: -------------- |++++++++#___| |_____+##__| |_____+#___| |_____+#_#_| |_###___+#_#_| |_#_#__#+++#_| |_###__#__+++| |_____###+| |____#_____+| --------------
1	0,0,11,0	The Maze -------------- |_____#___| |_____##__| |_____#___| |_____#_#_| |_###___#_#_| |_#_#__#___#_| |_###__#_____| |_____###_| |____#_____| -------------- Find a passage from start [x]: 0 Find a passage from start [y]: 0 To destination [x]: 11 To destination [y]: 0 A passage from [0,0] to [11,0] could be found! The passage through the maze is: -------------- |++++++++#__+| |_____+##_+| |_____+#__+| |_____+#_#+| |_###___+#_#+| |_#_#__#+++#+| |_###__#__+++| |_____###_| |____#_____| --------------	1	0,0,10,0	The Maze -------------- |_____#___| |_____##__| |_____#___| |_____#_#_| |_###___#_#_| |_#_#__#___#_| |_###__#_____| |_____###_| |____#_____| -------------- Find a passage from start [x]: 0 Find a passage from start [y]: 0 To destination [x]: 10 To destination [y]: 0 A passage from [0,0] to [10,0] could be found! The passage through the maze is: -------------- |++++++++#_+_| |_____+##+_| |_____+#++_| |_____+#+#_| |_###___+#+#_| |_#_#__#+++#_| |_###__#_____| |_____###_| |____#_____| --------------
2	8,0,11,8	The Maze -------------- |_____#___| |_____##__| |_____#___| |_____#_#_|	2	0,0,8,0	The Maze -------------- |_____#___| |_____##__| |_____#___| |_____#_#_|

			_###____#_#_
_#_#__#___#_
_###__#_____
________###_
_____#______	 -------------- Find a passage from start [x]: 8 Find a passage from start [y]: 0 To destination [x]: 11 To destination [y]: 8 A passage from [8,0] to [11,8] could not be found!				_###____#_#_
_#_#__#___#_
_###__#_____
________###_
_____#______	 -------------- Find a passage from start [x]: 0 Find a passage from start [y]: 0 To destination [x]: 8 To destination [y]: 0 A passage from [0,0] to [8,0] could not be found!
2	8,0,8,0	The Maze -------------- 	________#___
________##__
________#___
________#_#_
_###____#_#_
_#_#__#___#_
_###__#_____
________###_
_____#______	 -------------- Find a passage from start [x]: 8 Find a passage from start [y]: 0 To destination [x]: 8 To destination [y]: 0 A passage from [8,0] to [8,0] could not be found!	3	-1,0,0,0	The Maze -------------- 	________#___
________##__
________#___
________#_#_
_###____#_#_
_#_#__#___#_
_###__#_____
________###_
_____#______	 -------------- Find a passage from start [x]: -1 Find a passage from start [y]: 0 To destination [x]: 0 To destination [y]: 0 A passage from [-1,0] to [0,0] could not be found!
3	0,0,-1,0	The Maze -------------- 	________#___
________##__
________#___
________#_#_
_###____#_#_
_#_#__#___#_
_###__#_____
________###_
_____#______	 -------------- Find a passage from start [x]: 0 Find a passage from start [y]: 0 To destination [x]: -1 To destination [y]: 0 A passage from [0,0] to [-1,0] could not be found!	3	12,8,12,8	The Maze -------------- 	________#___
________##__
________#___
________#_#_
_###____#_#_
_#_#__#___#_
_###__#_____
________###_
_____#______	 -------------- Find a passage from start [x]: 12 Find a passage from start [y]: 8 To destination [x]: 12 To destination [y]: 8 A passage from [12,8] to [12,8] could not be found!
4	0,0,9,0	The Maze -------------- 	________#___
________##__
________#___
________#_#_
_###____#_#_
_#_#__#___#_
_###__#_____
________###_
_____#______	 -------------- Find a passage from start [x]: 0 Find a passage from start [y]: 0 To destination [x]: 9 To destination [y]: 0 A passage from [0,0] to [9,0] could not be found!	4	0,0,3,6	The Maze -------------- 	________#___
________##__
________#___
________#_#_
_###____#_#_
_#_#__#___#_
_###__#_____
________###_
_____#______	 -------------- Find a passage from start [x]: 0 Find a passage from start [y]: 0 To destination [x]: 3 To destination [y]: 6 A passage from [0,0] to [3,6] could not be found!
5	0.5, “abc”, “”	… Find a passage from start [x]: 0.5 Not a valid int, please try again: abc Not a valid int, please try again: Not a valid int, please try again:			

Question 2: My Little Pattern Matcher

a) The idea of my solution

First we initialize two arrays of the Strings for the testing: “TEXTS” and “PATTERNS”. Using loop we print out the results on the console. In the function “isMatching” we check given parameters for the null values. Then we check whether the given Strings are empty. If the pattern is not empty we continue our calculations with the “switch”. The algorithm is next:

- If the first symbol in pattern is equal to ‘?’, then recursively call isMatching for text and pattern without their first symbols.
- If the first symbol in pattern is equal to ‘.’ and the first symbol in text is an alphabet letter, then recursively call isMatching for text and pattern without their first symbols.
- If the first symbol in pattern is equal to ‘#’ and the first symbol in text is a digit, then recursively call isMatching for text and pattern without their first symbols.
- If the first symbol in pattern is equal to ‘*’ then recursively call isMatching for text and pattern without first symbol. If the result is false, do an action that described for the symbol ‘+’.
- If the first symbol in pattern is equal to ‘+’ then in loop for ‘i’ from the number one to the length of the text recursively call isMatching for the text without first ‘i’ symbols and pattern without first symbol.
- If the first symbol in pattern is equal to ‘$’ then in loop for ‘i’ from the number one to the length of the text, while previous symbol is a digit, recursively call isMatching for the text without first ‘i’ symbols and pattern without first symbol.
- Otherwise check whether first symbols from the text and from the pattern are equal. If they are, then recursively call isMatching for text and pattern without their first symbols and if they are not, then return false.

b) Source code

- Main.java

```
package uebung10.question2;

public class Main {
  private static final String[] TEXTS = { null, "", "", "", "abc", "abc",
      "abc", "abcd", "abc", "abc", "", "123", "123", "123", "123", "123.",
      "!Hugo12344!", "!Hugo12344!", "aaaaaaaaaaaaaaaaaaaababababababa",
      "43(699)108-934-62", "43(699)108-934-6", "office@pervasive.jku.at",
      "office@" };
  private static final String[] PATTERNS = { null, "", "abc", "***", "", "abc",
      "abcd", "abc", "*", "+", "+", "*#", "+#", "$", ".$", "$.", ".Hugo###$?!",
      "!Hugo###$?!", "*b*b*b*b*b*", "43(###)###-###-##", "43(###)###-###-##",
      "+@+", "+@+" };

  /** @param args - no arguments will evaluate */
  public static void main(String[] args) {
    for (int i = 0; i < TEXTS.length; i++)
      System.out.println("isMatching (\"" + TEXTS[i] + "\", \"" + PATTERNS[i]
          + "\"); => " + isMatching(TEXTS[i], PATTERNS[i]));
  }

  /** Checks whether the text matches to the pattern
   *
   * @param text - the text to check
   * @param pattern - the pattern to check
   * @return - true if the pattern matches */
  public static boolean isMatching(String text, String pattern) {
    // check whether the input is correct
    if (text == null || pattern == null)
      return false;
```

```
    if (pattern.length() == 0) {
      // if both Strings are empty, return true
      if (text.length() == 0)
        return true;
      return false;
    }
    // the text can be empty only if pattern is empty or contains star symbols only
    if (text.length() == 0 && pattern.charAt(0) != '*')
      return false;
    String patternSubstring = pattern.substring(1);
    // for every described symbol do a correspondent action
    switch (pattern.charAt(0)) {
    case '?':
      return isMatching(text.substring(1), patternSubstring);
    case '.':
      char c = text.charAt(0);
      if ((c < 'a' || c > 'z') && (c < 'A' || c > 'Z'))
        return false;
      return isMatching(text.substring(1), patternSubstring);
    case '#':
      if (text.charAt(0) < '0' || text.charAt(0) > '9')
        return false;
      return isMatching(text.substring(1), patternSubstring);
    case '*':
      if (isMatching(text, pattern.substring(1)))
        return true;
    case '+':
      for (int i = 1; i <= text.length(); i++)
        if (isMatching(text.substring(i), patternSubstring))
          return true;
      return false;
    case '$':
      char digit = text.charAt(0);
      for (int i = 1; i <= text.length() && digit >= '0' && digit <= '9'; digit = text
          .charAt(i++))
        if (isMatching(text.substring(i), patternSubstring))
          return true;
      return false;
    default:
      if (text.charAt(0) != pattern.charAt(0))
        return false;
      return isMatching(text.substring(1), patternSubstring);
    }
  }
}
```

c) Test plan

#	Description	Expected output
1	Check null values	**Return false**
2	Check a behavior with empty strings	**Return correspondent output described in the requirements**
3	Check alphabet symbols	**Return correspondent output described in the requirements**
4	Check other combinations	**Return correspondent output described in the requirements**

d) The output of the program

Output	Comments
isMatching ("null", "null"); => false isMatching ("", ""); => true isMatching ("", "abc"); => false isMatching ("", "***"); => true isMatching ("abc", ""); => false isMatching ("abc", "abc"); => true isMatching ("abc", "abcd"); => false isMatching ("abcd", "abc"); => false	**Test #1** **Test #2** **Test #3**

isMatching ("abc", "*"); => true isMatching ("abc", "+"); => true isMatching ("", "+"); => false isMatching ("123", "*#"); => true isMatching ("123", "+#"); => true isMatching ("123", "$"); => true isMatching ("123", ".$"); => false isMatching ("123.", "$."); => false isMatching ("!Hugo12344!", ".Hugo###$?!"); => false isMatching ("!Hugo12344!", "!Hugo###$?!"); => true isMatching ("aaaaaaaaaaaaaaaaaaaaababababababa", "*b*b*b*b*b*"); => true isMatching ("43(699)108-934-62", "43(###)###-###-##"); => true isMatching ("43(699)108-934-6", "43(###)###-###-##"); => false isMatching ("office@pervasive.jku.at", "+@+"); => true isMatching ("office@", "+@+"); => false	**Test #4**