Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Institute of Information Security and Dependability (KASTEL)
https://sdq.kastel.kit.edu/wiki/Programmieren/
programmieren-vorlesung@cs.kit.edu

Programming – Summer Semester 2024

Final task 1	20 points	Output:	13.08.2024, approx. 13:00
Version 1.0		Delivery start:	27.08.2024, 12:00
		Deadline:	11.09.2024, 06:00

Gender-fair language

If the generic masculine form has been chosen, this is done for better readability and to facilitate understanding of the task. Unless otherwise stated, information refers to representatives of all genders in the interests of equal treatment.

Plagiarism

Only independently prepared solutions will be accepted. Submitting other people's solutions, even partial solutions from third parties, from books, the Internet or other sources, is an attempt at deception and will result in a "failed" grade at any time (even retrospectively). Explicitly excluded from this are source code snippets from the lecture slides and from the suggested solutions from the practical sessions this semester. All resources used must be fully and accurately stated. Anything taken from the work of others, either unchanged or with modifications, must be clearly identified.

Students who disrupt the proper course of a performance assessment can be excluded from taking the performance assessment. Likewise, passing on parts of test cases or solutions, among other things, already constitutes a disruption of the proper course. These types of disruptions can also lead to exclusion from the performance assessment at any time. This expressly means that the score can also be reduced retrospectively.

Communication and current information

In our *FAQ*[1] you will find an overview of frequently asked questions and the corresponding answers to the “Programming” module. Please read these carefully before you

[1] https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ

Ask questions and check them regularly and independently for changes. Also note the information in the ILIAS Wiki[2].

In the*ILIAS forums*or on*Artemis*we occasionally publish important news. Any corrections to tasks will also be announced this way. Active monitoring of the forums is therefore required.

Check your mailbox*KIT email address*Check regularly for new emails. You will receive a summary of the corrections by email to this address. You can then enter all comments in the online submission system[3]view.

Editing instructions

Please note that successfully passing the mandatory tests is necessary for successfully submitting final task 1. Your submission will automatically be graded with zero points if any of the following rules are violated. You must first pass the mandatory tests before the other tests can be evaluated. Plan accordingly for your first submission attempt.

- Make sure that the program code compiles without errors.
- Only use*Java SE17*.
- Unless explicitly stated otherwise in an assignment, do not use any elements of the Java libraries. Except for the classjava.util.Scannerand all elements from the following packages:java.lang, java.io, java.util, java.util.regex, java.nio.file, java.nio.charset.
- Be careful not to create lines, methods and files that are too long. Your solutions must have a maximum line width of 140 characters.
- Follow all whitespace rules.
- Follow all rules regarding variable, method and package naming.
- Choose appropriate visibilities for your classes, methods and attributes.
- Do not use the**default**-Package.
- System.exit(), Runtime.exit()or similar may not be used.
- Follow the Javadoc documentation rules.
- Follow all other checkstyle rules.

The following processing instructions are relevant for the evaluation of your submission. However, your submission will be processed by the submission system*not*automatically rated with zero points if one of the following rules is violated. Please also refer to the evaluation criteria in the ILIAS Wiki.

[2]https://ilias.studium.kit.edu/goto.php?target=wiki_2368037_Hauptseite
[3]https://artemis.praktomat.cs.kit.edu/

- Do not include any personal information in your submissions other than your u-signature.
- Please note that your submissions will be assessed in terms of both object-oriented modeling and functionality. Follow the modeling guidelines in the ILIAS wiki.
- Program code must be written in English.
- Comment your code appropriately: as much as necessary, as little as possible.
- The comments should be written uniformly in English or German.
- Specify only your u-abbreviation in the Javadoc author tag.
- Choose meaningful names for all your identifiers.

Checkstyle

The online submission system automatically checks your source code during submission for compliance with the Checkstyle rules. There are specially marked rules for which the online submission system gives the submission zero points, as these rules must be complied with. Other rule violations can lead to points being deducted. You can and should check your source code for compliance with the rules during development. The programming wiki in ILIAS describes how Checkstyle can be used.

Delivery instructions

The online submission system will open on August 27, 2024, 12:00 p.m. Be sure to upload your files to the correct task in the submission system before the deadline of September 11, 2024, 6:00 a.m. Start submitting early to test your solution and use the forum to clarify any ambiguities. If you are submitting with Git,*must always*on theMain-Branch will be pushed.

- Enter your *.Java-Files for task A individually with the corresponding folder structure in the associated directory.

Reuse of solutions

If you reuse sample solutions from this semester for the final tasks or exercise sheets,*must* You enter "Programming Team" in the author tag of the corresponding classes. This is necessary to fulfill the checkstyle criteria.

Exam mode in Artemis

When you have finished a final assignment, you can submit it early using the Submit Early button. Once you submit a final assignment early, you will not be able to make any changes to your submission.

Task A: Campus Chaos

In this final task you should *Campus Chaos*, develop a board game. The game is a variation of the classic game *Maleficent*[4]. In summary, it is a game in which players try to move their pieces from a starting point to a destination point. In doing so, they can capture opposing pieces and move obstacles. The game ends as soon as a player has brought one of their pieces to the destination.

A.1 The game in detail

This section describes the rules and playing field of *Campus Chaos* described in more detail. There can be a variety of playing fields, designed for very different numbers of players.

A.1.1 Playing field

The playing field of *Campus Chaos* consists of fields that are connected to each other. All fields must be connected horizontally and/or vertically. Diagonally adjacent fields are not connected. A field can therefore have a maximum of four neighbors. In addition, there is always exactly one goal to which the players must move their pieces. To illustrate this, a playing field can be seen as a × Grid. Please note that the playing field must be connected, i.e. no fields may be isolated. For each 2x2 block, at least one field may not belong to the playing field. Except for the start fields, the target field and the special field, *Foresta* field must also be connected to at least two other fields. Finally, there must be at least as many free fields (fields to which a piece can be moved) as there are playing pieces.

A simple playing field could look like this (Table A.1):

	0	1	2	3	4	5	6	7	8	9	10
0						T					
1						P					
2						P					
3					P	P	P				
4					P		P				
5		P	P	P	P	P	P	P	P	P	
6		P			P		P			P	
7		P			P	P	P			P	
8	P	P	P	P	P		P	P	P	P	P
9	P			P				P			P
10	a			b				c			d

Table A.1: Example of a playing field (11×11 grids) without obstacles and without special fields

[4] https://de.wikipedia.org/wiki/Malefiz_(Game)

The numbers on the edge are for orientation only and are not used in the game. In this example it is an 11×11 Playing field for a maximum of four players. The starting positions of the players are marked with the lettersa, b, canddEach player has exactly one starting position. The goal is marked with aT (Target). The players move from the starting positions towards the target. The players can move on the fieldsP (Pathway). A player has won when he has moved one of his pieces exactly to the target square. There is exactly one target square.

A.1.2 Rules of the game

Each player always has five pieces. The game itself can be played with a maximum of 21 players ([AZ] minus the reserved letters (see current and following sections)). The players take turns one after the other in a fixed order (ascending alphabetically). If the player's starting position (player name as a lowercase letter) is free, he can bring a new piece into play; this is independent of the dice roll. In principle, starting squares may not be entered by rolling dice. However, moving over starting squares is permitted.

In each turn, a player rolls the die exactly once and moves exactly one of his pieces by the number rolled. The number rolled must always be moved in full. No dice point may be lost. It is also not permitted to move a piece only part of the number rolled or to move first forwards and then backwards within one turn. Other pieces and your own pieces may be skipped, whereby the skipped square is included. It may happen that no piece can be moved. Only in this case does the player miss a round and skip the turn. Otherwise, skipping turns is not permitted. In general, only one piece may be on each square. If a piece roll lands exactly on a square that is already occupied by another player's piece, it is beaten. A beaten piece can be brought back into play via the player's starting square at the beginning of a turn, just like the pieces at the start of the game. Your own pieces may not be beaten.

A.1.3 Special fields

To make the game more interesting, there are some extensions to the playing field. An example playing field with obstacles and special fields is shown in Table A.2 and explained in more detail below.

A.1.3.1 Blockages Firstly, there are blockades that players cannot jump over. These are marked with a smallOor large*O*(Obstacle). The meaning of the upper and lower case letters is explained in A.1.3.3. Since obstacles cannot be jumped over, there is a way to move them. If you move a figure exactly onto an obstacle, it is removed and can be placed somewhere else on the playing field. Obstacles cannot be placed on starting positions or the goal.

	0	1	2	3	4	5	6	7	8	9	10
0						T					
1						O					
2						p					
3					O	p	O				
4			e		p		p				
5		O	P	Z	p	p	p	Z	P	O	
6		P			P		P			P	
7		O			O	P	O			P	
8	O	P	P	O	P		P	O	P	P	O
9	P			P				P			P
10	a			b				c			d

Table A.2: Example of a playing field with obstacles and special fields

A.1.3.2 Protection zones On the other hand, there are protection zones in which the players' figures are safe. Only one figure can be in a protection zone at a time. Protection zones are marked with aZ (Zone). No blockades may be placed in a protection zone. Pieces in protection zones cannot be captured.

A.1.3.3 Village and forest All previous types of fields except the starting fields, the goal and the protection zones can be part of the*Village*The village is a continuous area of fields. Fields belonging to the village are marked with a smallp (Pathway) or small*O*(Obstacle). Fields that do not belong to the village are marked with a largeP (Pathway) or large*O* (Obstacle). If a figure is captured while it is in the village, it is moved to the forest (field*e*(Forest)). The forest is a field on which several game pieces can be located; however, it cannot be entered directly. There is a maximum of one such forest. Forest and village can only appear together in a game. The property of whether a field belongs to the village is tied to the position of the field. This means in particular that if an obstacle is moved from the village to outside the village, it no longer belongs to the village.

A.2 Implementation of the game

This section describes the implementation of the game. The game is to be implemented with a textual interface.

information

Notes on implementing this final task
Please note again the definition of the permitted packets at the beginning of the task. In particular, avoid using streams.

Since we perform automatic tests of your interactive user interface, the output must be exactly as specified. In particular, lowercase and uppercase letters as well as spaces and line breaks must match exactly. Only use the

Do not enter any additional information. For error messages, you can choose the English text freely, but it should make sense. However, every error message must be accompanied by Errors,and must not contain any special characters such as line breaks or umlauts.

Unless otherwise specified, input is always standard inputSystem.inUnless otherwise specified, output always uses the standard outputSystem.outFor error messages, the standard error output can optionally be used instead of the standard output.System.err Never reassign this standard input and output.

A.2.1 Representation of the example interactions

In example interactions, the symbol %> (percent sign and greater than sign followed by a space) represents the command line. The program name is freely chosen and does not have to be*Campus Chaos*The symbol > (greater than sign followed by a space) represents user input and is not part of the input itself. If [...] is written in an interaction, this means that further interactions have been skipped in order to address a specific part of the interaction. [...] is not an output of your game. Representations in angle brackets <>, such as < session_id>are also placeholders. A short explanation is inserted within the brackets for comprehension. Note that such placeholders are never part of the output of your program. Finally, there is the placeholder␣,which is used to represent a space in selected example interactions. This is also never part of the program's output. Please note that you should not copy the interactions from this PDF, as copying from PDFs can cause changes. Use the text files in Ilias instead and ask in the relevant forums if you have any questions about the format.

information

Notes on special characters in the PDF
It is best not to copy directly from the body text of the PDF, as depending on the PDF viewer, there may be problems with special characters or spaces. Use the boxes with the example interactions as a guide.

A.2.2 Quit command

In general, your program must always be started by entering the commandquitNote that this command is essential for testing your submission, as many of the testsquit at the end of a test sequence to terminate your program. Therefore, make sure that the command works in every situation. Please note again that you do not need to System.exit()or use other excluded features.

A.2.3 Program start

Your program will not be started with any arguments. If an argument is passed, an error message will be displayed and the program will be terminated.

Example interaction

```
1 %> java CampusChaos
2 Welcome to CampusChaos 2024. Enter 'help' for more details. [...]
3
```

A.2.4 Command: help

The commandhelpoutputs a short description of the commands line by line, which*available* Available means that the command could be executed in the current state of the program. This means that a die can only be thrown once per player per turn (ie within a player's turn, after the die has been thrown once, the command to throw the dice is hidden). Please note the descriptions in the following sections in particular.

The description of the commands is up to you, but should be in English, like all output from the program. The description should be sufficient to understand the commands. It must not contain any line breaks. The order of the commands is ascending (lexicographically according to the name of the command). The format is always <Command name>: <description>.

Example interaction

```
1 [...]
2 > help
3 [...]
4 help: Add some valuable description here. [...]
5
```

The command is always available.

A.2.5 Session Management

Your program should allow you to manage several games at the same time. A game is called *session*A session has a unique ID, which is calledsession_idThe ID is any string that matches the regular expression [a-zA-Z0-9]+There is no session when the program starts.

A.2.5.1 show session The commandshow sessionshows all available sessions. The sessions are displayed line by line. They are displayed in the order in which the sessions were created. They are displayed in the form <session_id> -> Players: <Alphabetically sorted list of players> | Map: <Path to map file> | Seed: <Dice seed>.This includes the defined number of players, the file that defines the playing field and the seed for the

The seed for the dice roll is optional and is only displayed if it has been defined. See also the commandstart session.

If a parameter for the command issession_idonly the session with this ID will be displayed. If the active session is to be displayed, this is indicated by a * after the session_iddisplayed.

Example interaction

```
1  [...]
2  > show session
3  MySession1* -> Players: A,B,C,D | Map: ./my-fancy-map.txt MySession2 ->
4  Players: A,B | Map: ./my-fancy-map.txt | Seed: 42
5  > show session MySession2
6  MySession2 -> Players: A,B | Map: ./my-fancy-map.txt | Seed: 42 [...]
7
```

The command is available if there is at least one session.

A.2.5.2 start session The commandstart sessionstarts a new session and thus a new game. The command has four parameters, each separated by a space:session_id, file_to_field, num_of_playersand optionallyseed.session_idis the ID of the session. An existing session must not be overwritten with this command.file_to_fieldis the path to a file that defines the playing field. A path must not contain spaces. number_of_playersis the number of players participating in the game. Note that the number of players must be at least 2. The maximum number of players is defined by the playing field. The players are represented in ascending alphabetical order, ie player 1 is
a,Player 2 isband so on. Starting squares of non-playing players remain starting squares and may not be entered by other players.seedis an optional parameter that defines the seed for the dice roll. The specification or lack of specification of the seed influences the functionality of the commandroll dice.

The command is always followed by the output of the complete file (line by line) (if the file exists). This allows the player to view the board before starting the game. Even if the board is invalid, the contents of the file are always output. If the session has been started successfully, the ID of the session is output. In addition, the new session is set as the active session. This means that all game commands are only applied to this session.

Example interaction

```
[...]
> start session MySession1 ./my-fancy-map.txt 4 [...]

MySession1
It's player A's turn.
> start session MySession2 ./my-fancy-map.txt 2 42 [...]

MySession2
[...]
```

The command is always available.

Input file: Playing field The input file for the playing field is a text file that defines the playing field. The file is structured so that each line represents a line of the playing field. Spaces are non-existent fields. Otherwise, the fields follow the rules and abbreviations described above. Please note again that the playing field must meet the properties described above. The file for Table A.2 is shown below.

File (visible spaces)

```
␣␣␣␣␣T
␣␣␣␣␣o
␣␣␣␣␣p
␣␣␣␣␣opo
␣␣f␣p␣p
␣OPZpppZPO
␣P␣␣P␣P␣␣P
␣O␣␣OPO␣␣P
OPPOP␣POPPO
P␣␣P␣␣␣P␣␣P
a␣␣b␣␣␣c␣␣d
```

A.2.5.3 switch session The commandswitch-sessionchanges the active session. The active session is the session to which all game commands are applied. The command has a parameter that specifies thesession_idof the session to which you want to switch. A successful session change is confirmed by the output of thesession_id.Switching from the current session to the current session is not allowed.

Example interaction

```
[...]
> switch session MySession1
MySession1
> switch session MySession2
MySession2
[...]
```

The command is available if the current session can be changed.

A.2.5.4 delete session The commanddelete sessiondeletes a session. The command has a parameter that specifies thesession_idof the session that is to be deleted. Successful deletion of the session is confirmed with the output of thesession_id.If the active session is deleted, no session will be active afterwards.

Example interaction

```
1 [...]
2 > delete session MySession1
3 MySession1
4 > delete session MySession2
5 MySession2
6 [...]
```

The command is available if there is at least one session.

A.2.6 Game commands

The game commands are always applied to the active session and thus affect the game in that session. Please note that the game commands can only be executed when a session is active.

A.2.6.1 Additional expenses of your system There are several situations in which your system needs to make additional outputs. Firstly, when a player has won. In this case, the output is:Player <player_id in uppercase> has won!On the other hand, it is issued when a player captures a piece:Player <player_id in uppercase> has hit Player <player_id in uppercase>.It is also announced that it is a player’s turn:It's player <player_id in uppercase>'s turn.

A.2.6.2 show The commandshowshows the current playing field. The playing field is displayed line by line. The format of the output is identical to the format of the input file, whereby the following changes are inherently important:

1. A playing piece has an ID, which is defined by the letter of the player and a consecutive number. This ID cannot be changed by the playing piece. This means that if it is beaten, it keeps this ID and returns to the end of the list of playing pieces that are not on the playing field.
2. The current player's pieces are represented by the number in the ID.
3. The other players’ pieces are represented by the capital letter of the respective player.
4. Fields occupied by game pieces are always represented by a letter or a number.

5. If there are game pieces in the forest, this is indicated by a largeFinstead of a smalle shown.
6. In case a player is currently moving an obstacle (i.e. has already moved but has not yet placed the obstacle), the obstacle will not be displayed on the playing field.

Example interaction (visible spaces)

```
1  [...]
2  > ␣show
3  ␣␣␣␣␣T
4  ␣␣␣␣␣o
5  ␣␣␣␣␣p
6  ␣␣␣␣␣opo
7  ␣␣f␣p␣p
8  ␣OPZpppZPO
9  ␣P␣␣P␣P␣␣P
10 ␣O␣␣OPO␣␣P
11 OPPOP␣POPPO
12 P␣␣P␣␣␣P␣␣P
13 a␣␣b␣␣␣c␣␣d
14 [...]
```

The command is always available.

A.2.6.3 current player The commandcurrentplayershows the current player. If the player has already rolled the dice, the number on the dice is shown. If the player has not yet rolled the dice, this is shown.

Example interaction

```
1  [...]
2  > current player
3  It's player A's turn. Dice Roll: ? [...]
4
5  > current player
6  It's player B's turn. Dice roll: 4 [...]
7
```

The command is available as long as there is no winner.

A.2.6.4 roll dice The commandroll dicerolls a die. The die has six sides and shows a number between 1 and 6. The number rolled is displayed. If the session was created without a seed, the command expects the parameterdice_roll,which indicates which number was rolled. If the session was created with a seed, the dice roll is determined by the seed and no parameter is expected.

Example interaction

```
1 [...]
2 > roll dice
3 4
4 [...]
5 > roll dice 6
6 6
7 [...]
```

The command is available as long as the player whose turn it is has not rolled yet that turn.

Seed for the dice roll If the session was created with a seed, the dice roll is determined by this seed. To do this, you create a pseudorandom number generator with the seed once when the session starts. The permissible range of the seed is determined by the constructor of the pseudorandom number generator. Use the constructorRandom(**long** seed)[5]to create a new random number generator using the*Seeds*to instantiate. Now always use the function Random :: nextInt(**int**bound)to generate a random number between 0 and 5. This random number increased by one gives the number rolled.

A.2.6.5 new figure The commandnew figurebrings a piece into play for the current player and places it on the player's starting position. All pieces are numbered before the game begins (starting at 1). This number remains the same throughout the game. Pieces are brought into play in ascending order. If a piece is captured, it is placed at the back of the queue. If successful, the ID of the piece is output (see commandshow).

Example interaction

```
1 [...]
2 > new figure
3 A1
4 [...]
```

The command is available as long as not all of a player's pieces are in play and the player's starting square is free.

A.2.6.6 move The commandmovemoves a figure of the current player. The first parameter to be specified is the figure's ID. This is followed by a non-empty list of pairs of distances and directions. These represent the path the figure should take. It must also be ensured that the figure moves along paths. The distance is a positive integer and the direction is one of the four cardinal directions:up, down, leftand right.Note that it is forbidden to move a piece over the same square multiple times within one turn (moving back and forth is forbidden). The format of the command ismove

[5]https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Random.html

<figure> (<distance> <direction>) ....If move is successful, nothing is spent. If no further actions are required from the player, it is the next player's turn.

Example interaction

```
1 [...]
2 > move A1 1 up 2 right 1 up It's
3 player B's turn.
4 [...]
```

The command is available if it has not been used this turn, has already been rolled, and no player has won yet.

A.2.6.7 move obstacle The commandmove obstaclemoves an obstacle that has just been reached by the player. The command has as parameters a list of pairs of steps and directions similar to the commandmove.As opposed tomoveThe list is limited to a maximum of two pairs, as an obstacle can be moved freely on the playing field and is not bound to the paths on the field. This means that the final position can be defined starting from the old position with a maximum of two relative movements. If the obstacle has been moved successfully, nothing is output. Then it is the next player's turn.

Example interaction

```
1 [...]
2 > move obstacle 20 up 3 right It's
3 player B's turn.
4 [...]
```

The command is only available if an obstacle has not yet been placed.

A.2.6.8 skip turn The commandskip turnskips the current player's turn. If executed successfully, the command does not output anything.

Example interaction

```
1 [...]
2 > skip turn
3 It's player B's turn. [...]
4
```

The command is available if it has not been used this turn, has already been rolled, and no player has won yet.

A.2.6.9 rematch The commandrematchrestarts the game with the same players and the same playing field.

Example interaction

```
[...]
> rematch
It's player A's turn. [...]

```

The command is only available when a player has won.

A.3 Example interaction

Example interaction

```
%> java CampusChaos
Welcome to CampusChaos 2024. Enter 'help' for more details.
> start session TestSession ./input/small_board.txt 2 PPaPPP
PPPcPP
P OPTPO P
PPbPPP PPPdPP
TestSession
It's player A's turn.
> show session
TestSession* -> Players: A,B | Map: ./input/small_board.txt
> roll dice 4
4
> new figure
A1
> move A1 3 right 1 down
> move obstacle 1 down
It's player B's turn.
> show
PPaPPP      PPPcPP
P APTPO           P
PPbPPO PPPdPP
> new figure
B1
> current player
It's player B's turn. Dice Roll: ?
> roll dice 3
3
> current player
It's player B's turn. Dice roll: 3
```

Example interaction

```
> move B1 3 left
> move obstacle 1 up 1 right It's
player A's turn.
> new figure
A2
> show
PP2PPP      PPPcPP
P 1OTPO           P
PPbPPB PPPdPP
> roll dice 2
2
> move A2 2 left It's
player B's turn.
> new figure
B2
> show
APaPPP      PPPcPP
P AOTPO           P
PP2PP1 PPdPP
> roll dice 1
1
> move B1 1 up
Player B has hit Player A. It's
player A's turn.
> roll dice 1
1
> move A2 1 down It's
player B's turn.
> roll dice 1
1
> move B1 1 left
> move obstacle 3 left 1 up It's
player A's turn.
> show
PPaOPP      PPPcPP
2 PBTPO           P
PPBPPP PPPdPP
> roll dice 3
3
> skip turn
It's player B's turn.
> roll dice 3
3
> move B2 3 right It's
player A's turn.
```

Example interaction

```
> show
PPaOPP      PPPcPP
2 PBTPO           P
PPbPPB PPPdPP
> roll dice 6
6
> move A2 1 down 5 right
Player A has hit Player B. It's
player B's turn.
> roll dice 2
2
> move B1 1 left 1 down Player
B has hit Player A. It's player
A's turn.
> roll dice 1
1
> new figure
A3
> show
PP3OPP      PPPcPP
P PPTPO           P
PPbPPB PPPdPP
> move A3 1 right
> move obstacle 5 right 1 down It's
player B's turn.
> roll dice 1
1
> new figure
B3
> move B1 1 up
It's player A's turn.
> roll dice 5
5
> move A3 2 right 1 down 2 right
Player A has won!
> show
PPaPPP      PPPcPP
P BP3OO           P
PPBPPP      PPPdPP
> quit
```