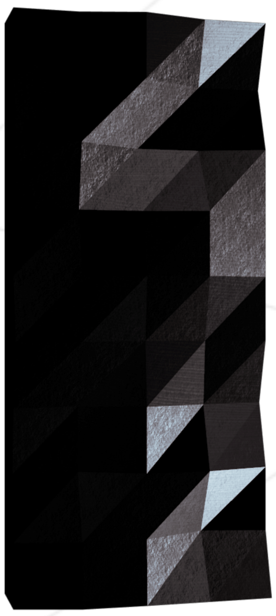


Rush 00 – Piscine Java

Console Game & Maven

Summary: Today you will implement quite a complicated game business process using Maven build tool

Contents

Chapter I

Foreword

Between 2000 and 2010, there were numerous issues in throttle controller software of a well-known automotive manufacturing company which resulted in 89 serious accidents.

The problem was not in the vehicle design but in low-quality software. It was a spaghetti code uncoverable by any testing methods.

Even NASA was engaged in the problem investigation.

Below is the number of defects found in the controller software:

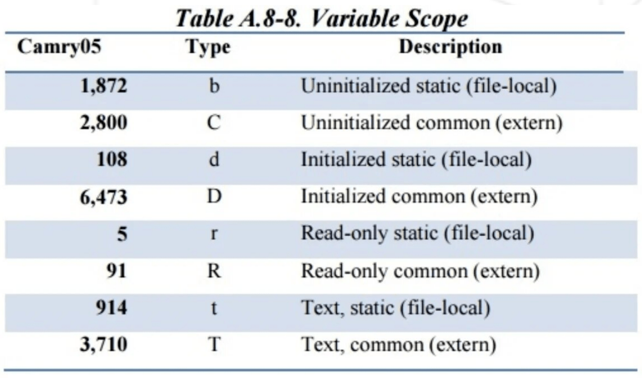


Table A.8-8. Variable Scope

Camry05	Type	Description
1,872	b	Uninitialized static (file-local)
2,800	C	Uninitialized common (extern)
108	d	Initialized static (file-local)
6,473	D	Initialized common (extern)
5	r	Read-only static (file-local)
91	R	Read-only common (extern)
914	t	Text, static (file-local)
3,710	T	Text, common (extern)

And this content is not safe to view:

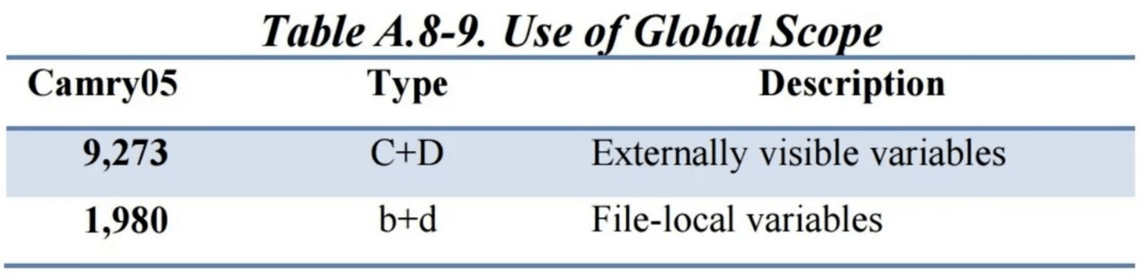


Table A.8-9. Use of Global Scope

Camry05	Type	Description
9,273	C+D	Externally visible variables
1,980	b+d	File-local variables

Chapter II

Instructions

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Now there is only one Java version for you, 1.8. Make sure that compiler and interpreter of this version are installed on your machine.
- You can use IDE to write and debug the source code.
- The code is read more often than written. Read carefully the document where code formatting rules are given. When performing each task, make sure you follow the generally accepted Oracle standards
- Comments are not allowed in the source code of your solution. They make it difficult to read the code.
- Pay attention to the permissions of your files and directories.
- To be assessed, your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. And one more thing. There's an answer to any question you may have on Stackoverflow. Learn how to ask questions correctly.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- Use "System.out" for output

- And may the Force be with you!
- Never leave that till tomorrow which you can do today ;)

Chapter III

Exercise 00 : Give up, You're Surrounded

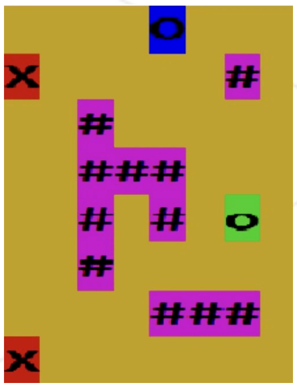
	Exercise 00
Surrender, You're Surrounded	
Turn-in directory : *ex*00/	
Files to turn in : Game-folder, ChaseLogic-folder	
Allowed functions : All	

Do you remember good old Java games? In the early 2000s, they were in each phone. Now Java developers design scalable enterprise systems, but at that time...

Your objective today is to get a bit nostalgic and implement a game where you run from artificial intelligence entities across a square field.

The program shall generate a random map with obstacles. Both player and its enemies are located on the map in a random manner. Each map element shall have a certain color.

Example of a generated map:



Designations:
o - position of a player (program user) on the map.
# - obstacle
x - enemy (artificial intelligence)

O - target point the player must get to before the enemies reach the player. The player is considered to have reached the target cell if they stepped on its position.

Game rules:
1. Each participant (player and enemies) may make one move. Then, it's another participant's turn. The enemy is considered to have reached the player if it can step on the player's position by making the current move.
2. Available movement directions are left, right, downward, and upward.
3. If an enemy is unable to move forward (there are obstacles or other enemies around them, or a map edge has been reached), the enemy skips a move.
4. The target point is an obstacle for an enemy.
5. If the player is unable to move forward (surrounded by obstacles, enemies, or has reached an edge of the map), the player loses the game.
6. The player loses if an enemy finds them before they reach the target point.
7. The player starts the game first.

Implementation requirements:
1. Field size, number of obstacles, and number of enemies are entered into the program using command-line parameters (their availability is guaranteed):

```
$ java -jar game.jar --enemiesCount=10 --wallsCount=10 --size=30 --profile=production
```

2. A check shall be made whether it is possible to put the specified number of enemies and obstacles on the map of given size. If input data is incorrect, the program shall throw an unchecked IllegalParametersException and shut down.
3. Enemies, obstacles, the player, and the target point are positioned on the field randomly.
4. When generating the map, enemies, the player, obstacles, and the target point shall not overlap.
5. In the beginning of the game, the map must be generated so that the player can reach the target point (the player must not be blocked by walls and map edge in starting position).
6. To make a move, the player shall enter a number in the console that corresponds to the movement direction A, W, D, S (left, upward, right, downward).
7. If the player is unable to make a move in the specified direction, another number (direction) shall be entered.
8. If the player understands in the beginning or middle of the game that the target point is unreachable, they shall end the game by entering 9 (player loses).
9. Once the player has made a move, it is its enemy's turn to make a move towards the player.
10. In the development mode, each enemy's step shall be confirmed by the player by entering 8.
11. Upon each a step of any participant, map must be redrawn in the console. In development mode, the map shall be displayed without updating the screen.
12. Pursuing algorithm shall take account of the target object location in each step.

Architecture requirements:
1. Two projects shall be implemented: Game (contains game logic, application entry point, output functionality, etc.) and ChaseLogic (contains pursuing algorithm implementation).

2. Both are mavenprojects, and ChaseLogic shall be added as a dependency to pom.xml inside Game.
3. Game.jar archive shall be portable: JCommander and JCDP must be directly included in the archive. At the same time, all libraries connected to the project shall be declared as maven-dependency. To build such archive, the following plugins shall be used.

It is also necessary to create a configuration file called application-production.properties. In this file, you will specify your application settings. The example of this file is shown below:

```
enemy.char = X
player.char = o
wall.char = #
goal.char = O
empty.char=
enemy.color = RED
player.color  = GREEN
wall.color  = MAGENTA
goal.color  = BLUE
empty.color = YELLOW
```

This configuration file will be located in resources folder of the launched jar archive.

In addition to that, application-dev.properties file should be implemented. Structure of this file is similar to that of application.properties. Here, you can specify parameters for distinguishing application startup in development mode (for example, different colors/characters for map components).

It is necessary to keep in mind that the program may also be started in other modes. For this purpose, the respective properties file can be added to the source project, and the mode itself is passed via --profile parameter.