Assignment 3

SE1 Proseminar, Winter 2007/2008

Stefan Resmerita
stefan.resmerita@cs.uni-salzburg.at

The factory layout is given as a rectangular grid, where grid lines are pathways on which robots can travel. The distance between any two adjacent vertices in the grid is always the same and it is denoted by Δs. A machine is represented as a rectangle, as shown in Figure 1. Every machine has a name. The dimensions of a rectangle (length, width) are multiples of Δs and the corners are placed at grid vertices.

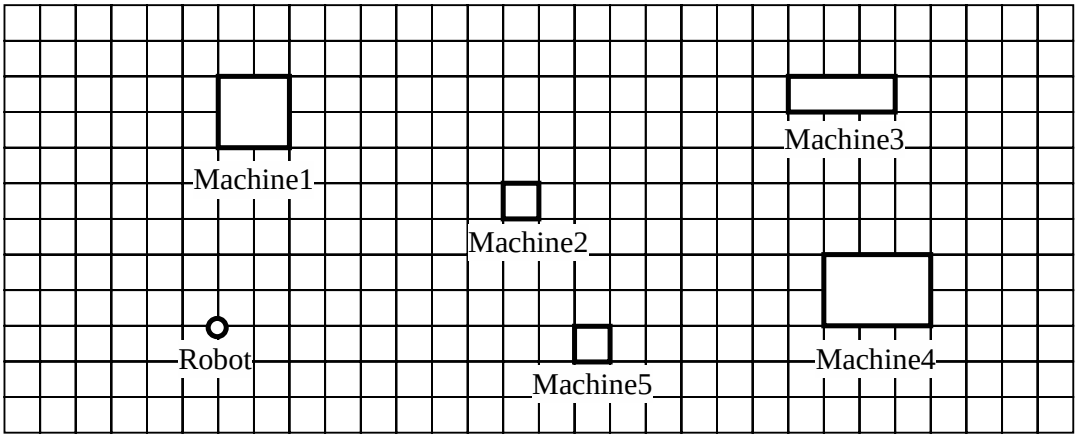

Figure 1. Example of factory layout

We assume that robot dimensions are relatively small with respect to Δs. Machines have fixed locations, while robots may enter and exit the system at any grid vertex on the margins of the domain.

A *floor layout* of this system is a specification of:

- A rectangular grid.
- A list of machines and their placements in the grid.

A floor layout is called *valid* if there exists at least one free pathway (gridline) between any two machines. Examples of invalid configurations are given in Figure 2.

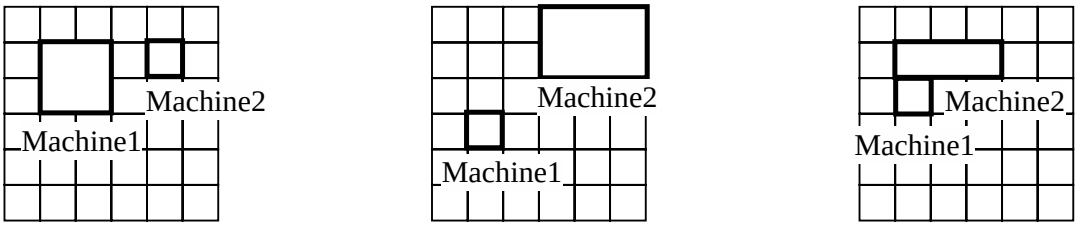

Figure 2. Examples of invalid configurations

Exercise 1

Implement a configuration system, where at least the floor layout is specified in a configuration file. Design the format of the configuration file in any way you want.
The application reads the configuration file and uses reflection to load the given settings. The settings are stored within a configuration class. Therefore, each setting should be described by a pair (*field*, *value*), where the field belongs to the configuration class. These fields are globally available, and each class of the application is responsible for initializing its fields with the values from the configuration class.

The application accepts user commands only if the floor layout is valid. Otherwise, it exits with an error message.

The configuration may contain other parameters, such as, for example, the number of available robots (of each type).

Exercise 2

Use the hook method principle to adapt your application to one of the following situations:

A. Every user command must be guarded by a password. Thus, whenever the user sends a command, it is asked for the password. The command is executed only if the password is correct. The user should be able to change the password with the command

   `SetPassword <password>`

   where **`<password>`** is a string containing only alphanumeric characters, without spaces. The default value of the password (valid upon installation of the application, before the first **`SetPassword`** command) is *robot*. The password must be persistent between executions of the application. Thus, every time the application starts up, it uses the password set by the last **`SetPassword`** command, or the default password if no (successful) **`SetPassword`** command has been issued in previous executions.

B. Same as above, with the only exception that the application asks for the password at startup, not after every command. If the supplied password is correct, then the application accepts user commands (without asking for the password after every command). Otherwise, the application terminates with an error message.

C. No password is used at all.

The checking package follows. Please make sure to implement the interface in this package.

```
package checking;

public interface CheckerInterface {


	/*
	 * Starts operation of the system.
	 */
	public void start();


	/*
	 * Terminates system operation: releases all resources.
	 * Make sure that this method returns (no System.exit here).
	 */
	public void stop();


	/*
	 * Returns the machine names in a String array. Returns null if no
	 * machine exists.
	 *
	 */
	public String[] getMachineNames();


	/*
	 * Determines execution of the command given by the string s,
	 * as if s were entered by the operator at the console. Returns
	 * true if the command execution completed successfully, false
	 * otherwise.
	 * Example: s="SetPassword r4ti5o".
	 */
	public boolean sendCommand(String s);


	/*
	 * Returns the student numbers of the authors. Each number is
	 * represented as a String
	 */
	public String[] getStudentIds();

}
```

```
package checking;

public class CheckerImplement implements CheckerInterface{

/* Place your implementation here */

}
```