Hand-in 5: PS10

|  |  |
| --- | --- |
| **GROUP 6** | |
| 19994567 | Tjip Pasma |
| 20119885 | Steffen Lind-Holt Poulsen |
| 201200077 | Torben Markussen |

21 May 2012

**Table of contents**

[1 Introduction 2](#_Toc325056302)

[2 UML / Package Diagrams before changes 3](#_Toc325056303)

[2.1 Class diagram 3](#_Toc325056304)

[2.2 Package diagram 4](#_Toc325056305)

[2.3 Scenario for user buying ticket: 4](#_Toc325056306)

[3 PS10 using RMI 5](#_Toc325056307)

# Introduction

This report concerns the exercise PS10.

# UML / Package Diagrams before changes

Diagrams are created using Enterprise Architect.

## Class diagram

Only public attributes/methods are visible in this class diagram.



## Package diagram



## Scenario for user buying ticket:

Scenario:  
*A user of pay station 1 enters four 25 cent coins and presses 'Buy' to receive a receipt. As the sale is processed the two monitor applications updates their displays with the proper total earning and number of vacant parking lots.*



# PS10 using RMI

The class diagram below shows the changes made to support an RMI implementation of the paystation system.



Important changes are:

* StatusObservable / StatusListener / StatusEvent are placed in a “common” package as they are shared classes (defines paystation monitoring protocol) between client and server.
* Implementation of the interface StatusObservable is moved from PayStation to PayStationImpl to avoid exposing the PayStation on RMI.
* StatusObservable / StatusListener are exported by reference
* Instances of StatusEvent are in reality immutable object and is exported by value.

# Production and test code

The production and the test code is zipped in the file code5.zip.

The Build.xml has the following targets:

runGUI : Will start 4 Paystation servers

runMonitor : Will start a monitor

run3Monitor : Will Start 3 monitors

When a monitor is started, it will search the rmiregistry every 2nd second for new Paystation servers. This means that it does not matter in what order you create monitors and server.

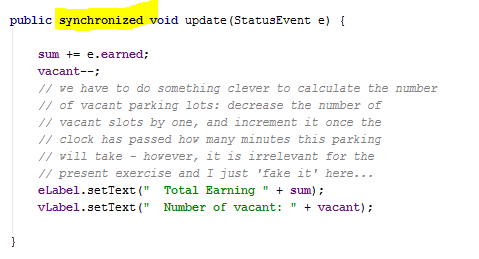
# Concurrency

There are two places in the program, where concurrency can be an issue:

1. When the client gets an event from the Paystation server

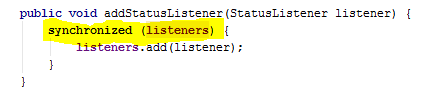
To make sure that the sum and vacant variables is updated correctly, the update method is synchronized.

To improve this even further, the status event could be added to a synchronized Queue, and processed by a separate thread. This would make the update method as fast as possible, and minimize the time different servers were waiting for other servers to finish.

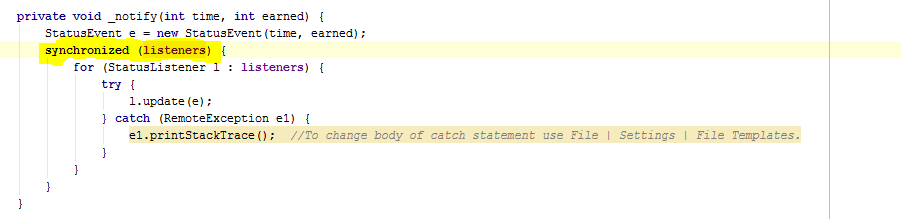


2. When a client is added to the servers listeners

To make sure that two listeners is not added at the same time, the listeners is synchronized, when adding new listeners to it.



3. When the server notifies the listeners

To make sure that a new listener is not added while iterating through the list of listeners, the whole for loop is synchronized. 

You could argue that the l.update is risky including in the synchronizing, because you do not know how long time a listener uses to update, and this will lock the listeners during the whole update. This could result in the server wanting to update the listeners faster than the listeners could process the update events, and would eventually lead to a stack overflow.

This is not likely to happen, because the update method from the server is triggered from human interaction (The buyers of the parking tickets), never the less, it would be better instead to make a local copy of the listeners, and iterating through the local list. Only the copying of the list then needs to be synchronized. If a listener is added while the server iterates through the list, the new listener is not updated, until the next update.