March, 2016

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Laboratory Work – Assignment 3

Thread management application

Name: *Popa Florin*

Group: *30425*

Table of Contents

1. Introduction

1.1 Problem specification

Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients' waiting time.

Description

Queues are commonly seen both in real world and in the models. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue based systems is interested in minimizing the time amount its "clients" are waiting in queues. One way to minimize the waiting time is to add more servers, i.e. more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the supplier. When a new server is added the waiting clients will be evenly distributed to all current available queues.

The application should simulate a series of clients arriving for service, entering queues, waiting, being served and finally leaving the queue. It tracks the time the clients spend waiting in queues and outputs the average waiting time. To calculate waiting time we need to know the arrival time, finish time and service time. The arrival time and the service time depend on the individual clients – when they show up and how much service they need. The finish time depends on the number of queues, the number of other clients in the queue and their service needs.

Input data:

- Minimum and maximum interval of arriving time between clients; - Minimum and maximum service time;

- Number of queues;

- Simulation interval;

- Other information you may consider necessary;

Minimal output:

- Average of waiting time, service time and empty queue time for 1, 2 and 3 queues for the simulation interval and for a specified interval;

- Log of events and main system data;

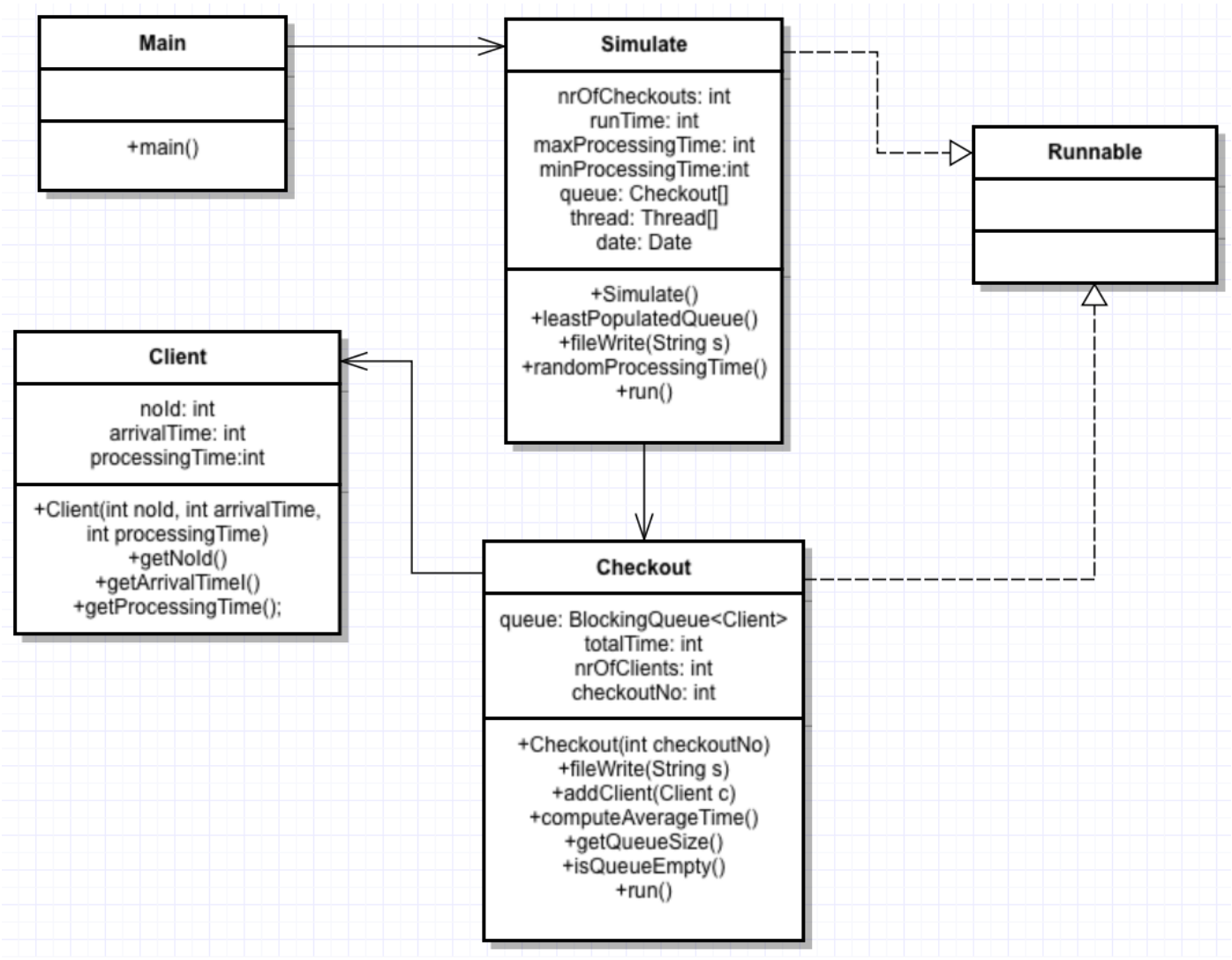
- Queue evolution;

- Peak hour for the simulation interval;

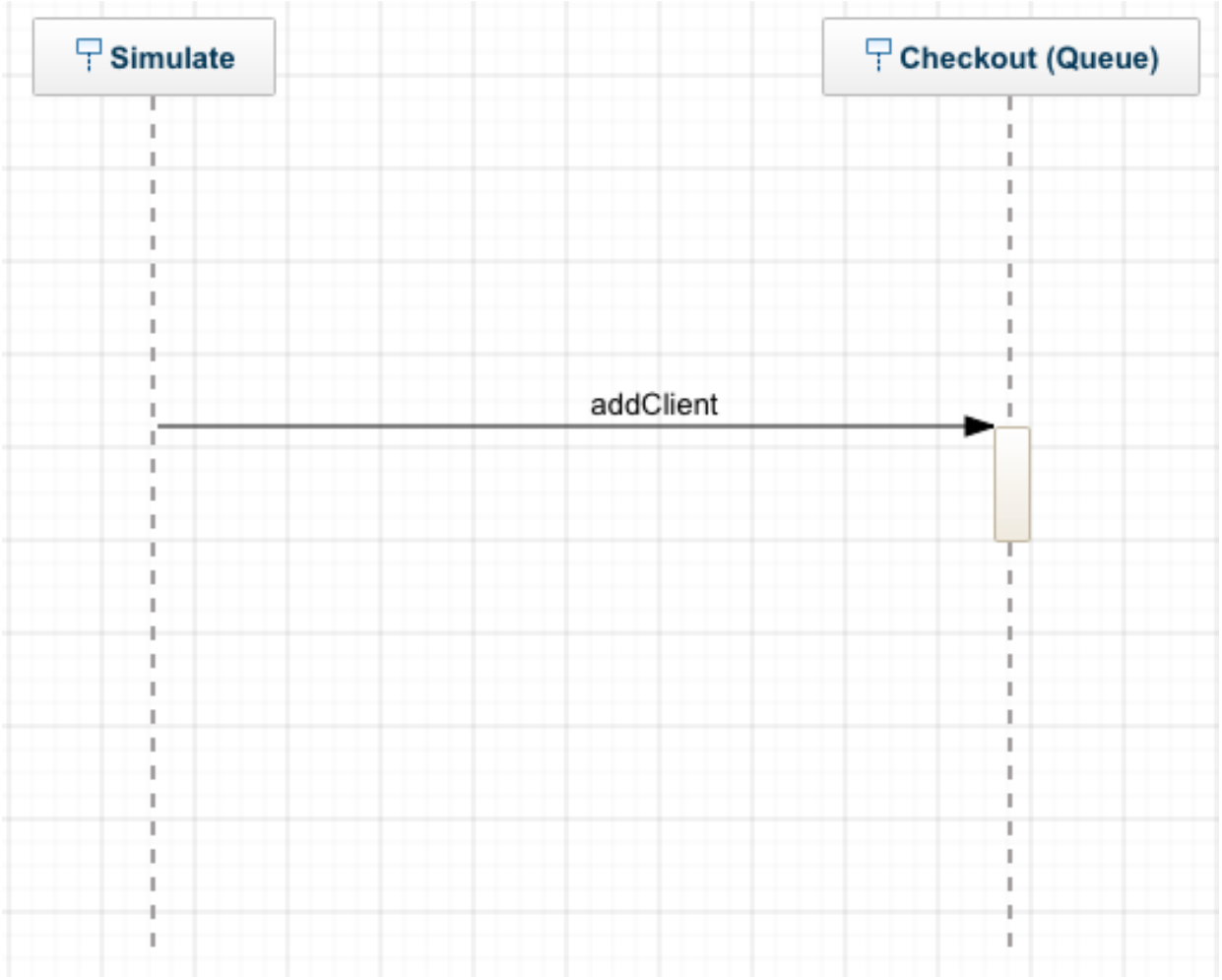
2. Design

2.1 UML diagrams

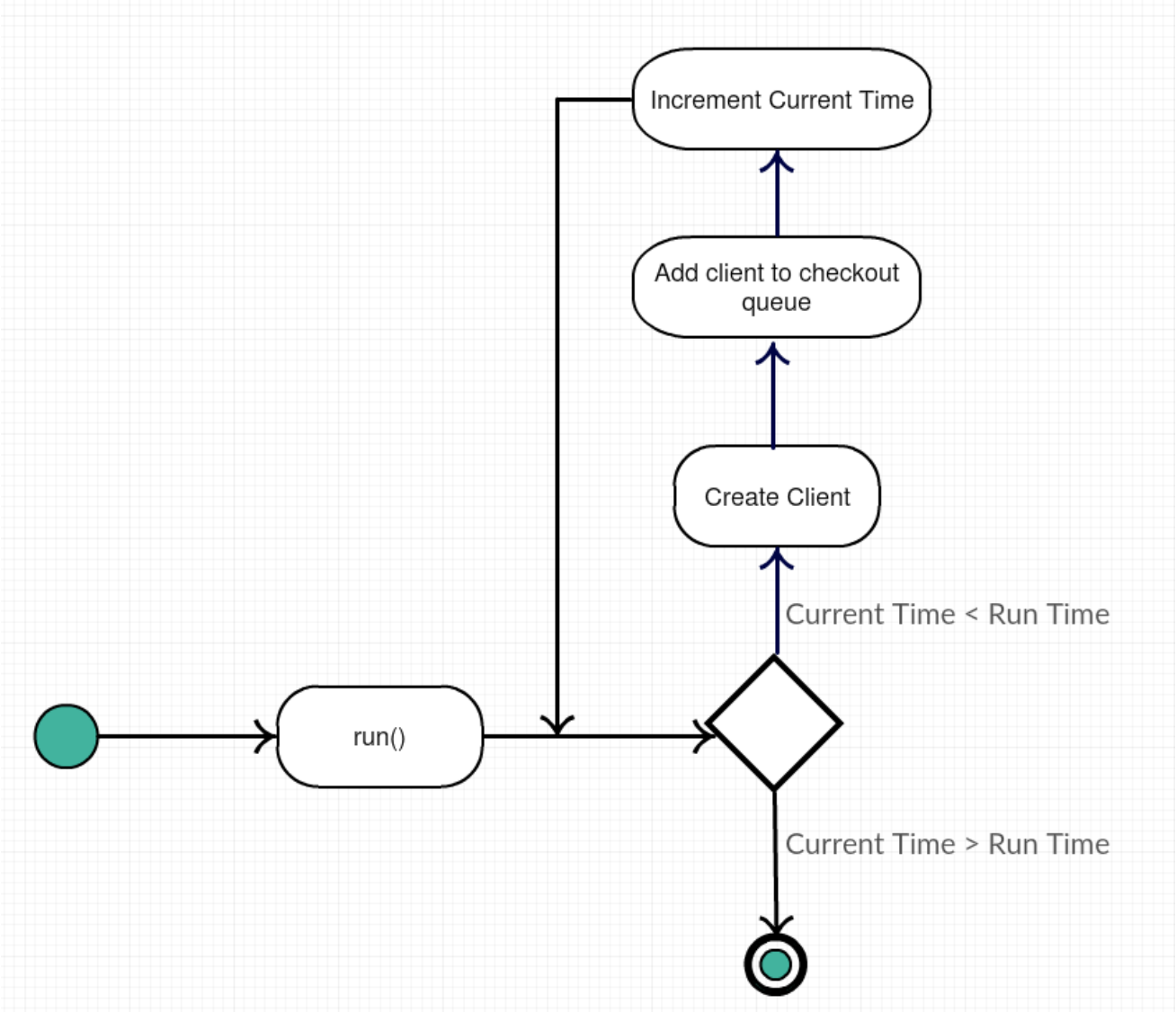
a) Class Diagram



b) Sequence diagram



c) Activity diagram



3. Packages and Interfaces

A Java package is a mechanism for organizing Java classes into namespaces. Java packages can be stored in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality. A package provides a unique namespace for the types it contains. Classes in the same package can access each other's package-access members.

A package allows a developer to group classes (and interfaces) together. These classes will all be related in some way – they might all have to do with a specific application or perform a specific set of tasks.

For this application the following packages are imported, each of them having a certain role for the proper working of the application. We import them in the GUI Class (most of them relate to the user interface properties):

1) import java.awt: Contains all of the classes for creating user interfaces and for painting graphics and images. A user interface object such as a button or a scrollbar is called, in AWT terminology, a component. The Component class is the root of all AWT components.

a. java.awt.Font: The Font class represents fonts, which are used to render text in a visible way. A font provides the information needed to map sequences of characters to sequences of glyphs and to render sequences of glyphs on Graphics and Component objects.

2) import java.awt.event : Used in interacting with input devices such as the mouse and keyboard

a) java.awt.event.ActionEvent: A semantic event which indicates that a component-defined action occurred. This high-level event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed). The event is passed to every ActionListener object that registered to receive such events using the component's addActionListener method.

b) java.awt.event.ActionListener: The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

3) import javax.swing: Swing is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes (JFC) – an API for providing a graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

a) javax.swing.JButton: Buttons can be configured, and to some degree controlled, by Actions. Using an Action with a button has many benefits beyond directly configuring a button.

b) javax.swing.JFrame: An extended version of java.awt.Frame that adds support for the JFC/Swing component architecture.

c) javax.swing.JLabel: A display area for a short text string or an image, or both. A label does not react to input events. As a result, it cannot get the keyboard focus. A label can, however, display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it.

A JLabel object can display either text, an image, or both. You can specify where in the label's display area the label's contents are aligned by setting the vertical and horizontal alignment. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

d) javax.swing.JPanel: The JPanel class provides general-purpose containers for lightweight components. By default, panels do not add colors to anything except their own background; however, you can easily add borders to them and otherwise customize their painting.

In many types of look and feel, panels are opaque by default. Opaque panels work well as content panes and can help with painting efficiently, as described in Using Top-Level Containers. You can change a panel's transparency by invoking the setOpaque method. A transparent panel draws no background, so that any components underneath show through.

e) javax.swing.JTable: The JTable is used to display and edit regular two-dimensional tables of cells. The JTable has many facilities that make it possible to customize its rendering and editing but provides defaults for these features so that simple tables can be set up easily.

4. Conclusions

Developing this application was good challenge. It was a very good practice as a laboratory assignment. Of course, many features could be implemented in future versions, such as providing the user with the possibility of creating a new account, changing the data structure with an SQL database, add graphic images to improve esthetics, etc.

5. Bibliography

- https://docs.oracle.com/
- http://stackoverflow.com
- https://wiki.eclipse.org