Event Registration Desktop Application

1 Required Software Engineering Tools

- Eclipse Mars Modeling Tools (http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/mars1)
- Umple Eclipse plugin 1.22.0.5146 (or later) (http://cruise.eecs.uottawa.ca/umpleonline/download_eclipse_umple_plugin.shtml)
- Umple Online (http://cruise.eecs.uottawa.ca/umpleonline/)
- JUnit 4 (included with Eclipse)
- XStream libraries (provided in myCourses)
- JDatePicker library (provided in myCourses)

Go to the above link and download the Eclipse Modeling Tools for your operating system.

Go to the above link and download the Umple Eclipse plugin by clicking on the "The latest official releases of Umple jars at GitHub" button in section B.1. Then, click on "Releases" at the top left of the GitHub page, because the latest version 1.23.0 does not yet include the Eclipse plugin. Scroll down to release 1.22.0.5146 and download *cruise.umple.eclipse_1.22.0.5146.jar*.

Install Eclipse and add the Umple jar into the *plugins* folder of Eclipse.

2 Description of Event Registration Desktop Application

Typically, this description is elicited from stakeholders (e.g., potential customers). However, for our purposes, we will assume that it is provided as follows:

- The Event Registration application shall provide the ability to add a participant by specifying the participant's name.
- The Event Registration application shall provide the ability to add an event by specifying the event's name, date, start time, and end time.
- The Event Registration application shall provide the ability to register a participant to an event.

3 Domain Model

See *EventRegistrationDesktop-DomainModel.html* in the *D03 Domain Model.zip* file to define the domain model with Umple Online and generate Java code with the Umple Eclipse plugin.

Note that Umple Online is useful to visualize the domain model with the textual specification at the same time. However if the server is overloaded, simply switch right away to the Umple Eclipse plugin and enter the textual specification there.

4 Persistence

You first need to add the 3 XStream JARs (*xstream-1.4.7.jar*, *xmlpull-1.1.3.1.jar*, *xpp3_min-1.1.4c.jar*) to your *lib* folder and to your build path. In addition, add the following class to your *src* folder.

```
package ca.mcgill.ecse321.eventregistration.persistence;

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import com.thoughtworks.xstream.XStream;

public class PersistenceXStream {

	private static XStream xstream = new XStream();
	private static String filename = "data.xml";

	public static boolean saveToXMLwithXStream(Object obj) {
		xstream.setMode(XStream.ID_REFERENCES);
		String xml = xstream.toXML(obj); // save our xml file

		try {
			FileWriter writer = new FileWriter(filename);
			writer.write(xml);
			writer.close();
			return true;
		} catch (IOException e) {
			e.printStackTrace();
			return false;
		}
	}

	public static Object loadFromXMLwithXStream() {
		xstream.setMode(XStream.ID_REFERENCES);
		try {
			FileReader fileReader = new FileReader(filename); // load our xml file
			return xstream.fromXML(fileReader);
		} catch (IOException e) {
			e.printStackTrace();
			return null;
		}
	}

	public static void setAlias(String xmlTagName, Class<?> className) {
		xstream.alias(xmlTagName, className);
	}

	public static void setFilename(String fn) {
		filename = fn;
	}

}
```

The above class is provided because you can easily find the code for writing to and reading from an XML file with XStream online. However, it is still necessary to test the persistence functionality of this class. See *EventRegistrationDesktop-TestPersistence.html* in the *D04 Test Persistence.zip* file to test the persistence functionality with JUnit 4. The test creates instances of classes in the domain model and

saves them to and loads them from an XML file using XStream. This requires the JUnit 4 library to be added to your build path.

5 Controller

See *EventRegistrationDesktop-Controller.html* in the *D05 Controller.zip* file for how to use the Test-Driven Development paradigm to implement one test for one method of the controller of the Event Registration Desktop Application and then implement this method of the controller.

6 Controller: Input Validation

See *EventRegistrationDesktop-ControllerInputValidation.html* in the *D06 Controller Input Validation.zip* file for how to use the Test-Driven Development paradigm to implement further tests for the same method as in Section 1 that now also take invalid inputs into account and then add input validation to this method of the controller.

7 View

See *EventRegistrationDesktop-View.html* in the *D07 View.zip* file for how to implement the User Interface of the Event Registration Desktop Application with the help of the Java Swing Framework.

First implement a User Interface that allows a participant to be added as shown in Figure 1. Then, add error handling as shown in Figure 2.

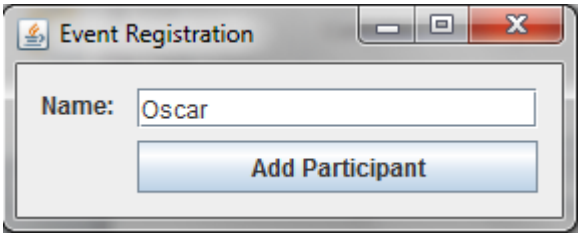

Figure 1 Add Participant

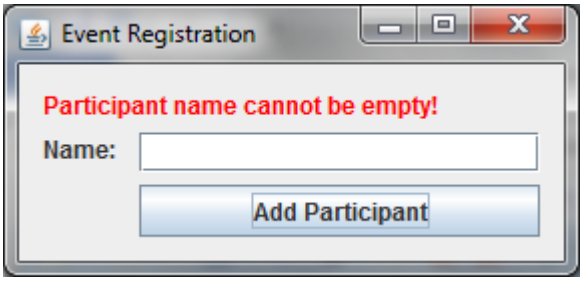

Figure 2 Add Participant Error Handling

8 Controller (Remaining Methods)

See *EventRegistrationDesktop-ControllerPart2.html* in the *D08 Controller Part 2.zip* file for how to use the Test-Driven Development paradigm to implement the "everything is ok" scenarios for the remaining methods of the controller of the Event Registration Desktop Application.

9 Controller: Input Validation (Remaining Methods)

See *EventRegistrationDesktop-ControllerInputValidationPart2.html* in the *D09 Controller Input Validation Part 2.zip* file for how to use the Test-Driven Development paradigm to implement the scenarios with invalid input for the remaining methods of the controller of the Event Registration Desktop Application.

10 View (Remaining Functionality)

See *EventRegistrationDesktop-ViewPart2.html* in the *D10 View Part 2.zip* file for how to implement the User Interface of the remaining functionality of the Event Registration Desktop Application with the help of the Java Swing Framework. You first need to add *jdatepicker-1.3.4.jar* to your *lib* folder and to your build path. In addition, add the following class (which can easily be found online) to your *src* folder. The JAR file and the class are needed for a UI element for selecting dates.

```
package ca.mcgill.ecse321.eventregistration.view;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;

import javax.swing.JFormattedTextField.AbstractFormatter;

public class DateLabelFormatter extends AbstractFormatter {

    private static final long serialVersionUID = -2169252224419341678L;

    private String datePattern = "yyyy-MM-dd";
    private SimpleDateFormat dateFormatter = new SimpleDateFormat(datePattern);

    @Override
    public Object stringToValue(String text) throws ParseException {
        return dateFormatter.parseObject(text);
    }

    @Override
    public String valueToString(Object value) throws ParseException {
        if (value != null) {
            Calendar cal = (Calendar) value;
            return dateFormatter.format(cal.getTime());
        }

        return "";
    }

}
```

First, add the functionality for adding events to the User Interface as shown in Figure 3. Then, add the functionality for registering a participant to an event as shown in Figure 4.

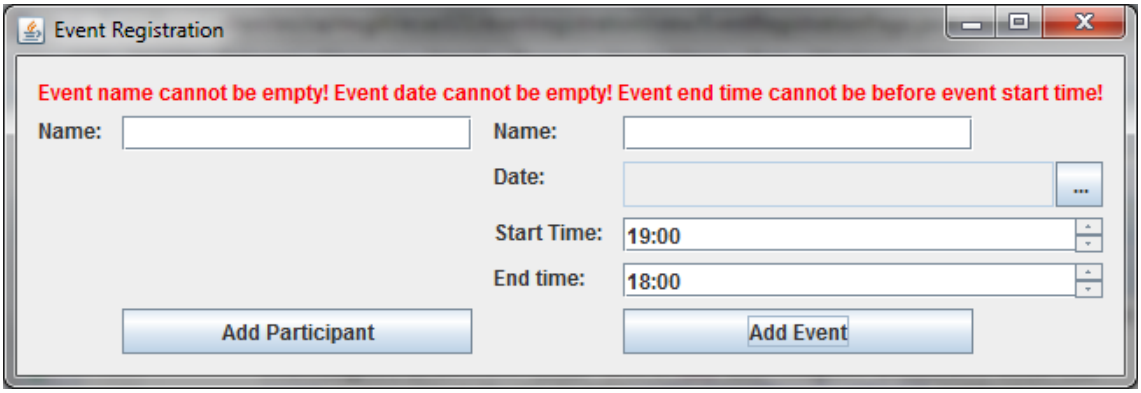

Figure 3 Add Participant and Add Event

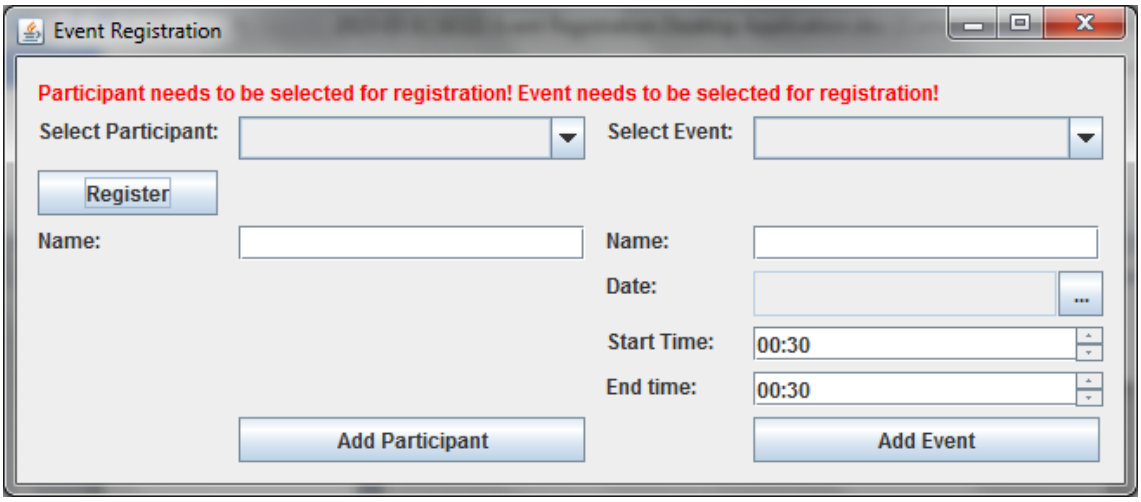

Figure 4 Add Participant, Add Event, and Register Participant to Event

Submission

This assignment is to be done in teams of TWO students. You are required to sign up to one of the assignment groups in myCourses. Your team is required to hand in a ***single zip file*** of the Eclipse project with your implementation by ***Friday, September 30th, 2016 at 23:59***. To create the zip file, use the *Export* feature of Eclipse to create an *Archive File* (Export – General – Archive File) of the project. If you realize that you need to make changes to your submission, do not resubmit only the file(s) that have changed, but rather resubmit another complete zip file.

Note that this is only part 1 of 3 parts. The second part of assignment #1 will be posted on September 15th, and the third part of assignment #1 will be posted on September 22nd. Each part is about the same amount of work and all parts are due on Friday, September 30th, 2016. Do not wait until the very end to complete this assignment. Start early.

Each team member must make contributions to the assignment. A team member who does not contribute to the assignment receives a mark of 0 for the assignment. A team member may optionally email a confidential statement of work to the instructor before the due date of the assignment. A statement of work first lists in point form the parts of the assignment to which the team member contributed. In addition, the statement of work also describes whether the work load was distributed fairly evenly among the team members. A statement of work may be used to adjust the mark of a team member who is not contributing sufficiently to the assignment. It is not necessary to send a statement of work, if a team distributed the work for the assignment fairly evenly and each team member contributed sufficiently.

Marking Scheme

Part of Assignment	***Marks***
Domain Model in Umple	15
Use of Code Generated from Domain Model	5
Adherence to Model-View-Controller Pattern	20

JUnit Tests	10
Implementation of Functionality:	40
a) Add Participant	10/40
b) Add Event	10/40
c) Register	10/40
d) Persistence	10/40
Implementation of Validation Checks for:	10
a) Add Participant	2/10
b) Add Event	4/10
c) Register	4/10
Total Marks:	100
The total mark may be adjusted based on the actual contributions of a team member to the assignment.	