Project #4 – Corrected

Computer Science 2334
Spring 2010

User Request:

"Create a Olympics Information System with a Graphical User Interface."

Milestones:

1. Create appropriate classes, complete with variables and methods, to handle the application data on Olympics as described below under model. *15 points*
2. Add a class, complete with data and methods, to the model to allow the model to correctly interact with the controller and the views. *10 points*
3. Create appropriate classes, complete with variables and methods, for the views described below. *25 points*
4. Create an appropriate class, complete with variables and methods, for the controller as described below. *25 points*

- ► Develop and use a proper design. *15 points*
- ► Use proper documentation and formatting. *10 points*

Description:

An important skill in software design is extending the work you have done previously. For this project you will rework Project #3 in order to handle information pertaining to Olympic Games and allow users to view and manipulate that information graphically. This project will be organized around the model, view, controller (MVC) design pattern which gives us a way to organize code involving graphical user interfaces (GUIs). In particular, you will create a single model to hold the data, several views to display and manipulate different aspects of the data, and one controller to moderate between user gestures and the model and views. For this program you may reuse some of the classes that you developed for your previous projects, although you are not required to do so. Note that much of the code you write for this program could be reused in more complex applications.

Model:

You will create a model class called "**OlympicsModel**." Models in the version of the MVC design pattern we have used in class contain data and methods for the application objects being modeled (which are all related to the Olympics in this project) as well as data and methods to allow the model to interact with views. Your model class will follow this version of the MVC design pattern.

For the application objects, you will create a class to represent a single Olympics. This Olympics will consist of a name (such as "XXI Olympic Winter Games"), a year (such as "2010"), a host city location (such as "Vancouver, British Columbia, Canada"), a list of sporting events, and a list of participating national teams.

In turn, each event will consist of an Olympics at which this event took place (see above), a name (such as "Women's Downhill Skiing"), a team size (which would be one for an individual event such as downhill skiing, two for a pairs event such as ice dancing, and larger for team sports such as hockey), and a list of participation records.

Similarly, each national team will consist of an Olympics at which this event took place (see above), a country name (such as "United States of America"), and a roster of athletes who are on the team.

Each athlete will be a person with a name, belong to a particular national team, and have a list of participation records.

Each participation record will consist of an event (see above), a competitive unit (a single athlete for an individual event, a pair of athletes for a pairs event, or a team of athletes for a team event), and the medal won by that competitive unit (if any).

All of the data for this model will enter the system through user input using the input views described below, after which time it may be saved to a file and reloaded as desired. This saving and loading will be in a "native" file format, rather than a text file. That is to say, it will use object I/O rather than text I/O. The model, therefore, must be able to save and load itself using object I/O.

To interact with views, the **OlympicsModel** class will have variables and methods akin to those from the examples we have seen in MVC lectures and labs. In particular, when new information is entered into the system through the input view and added to the model, any relevant display views should be notified so that they may update themselves to reflect the new data.

Views:

Producing views of information can be very useful to users. Therefore your program will create and maintain several views of the data, as described below.

Home View:

The *Home View* will be displayed as soon as the program is started. It will initially consist of a multi-color image of the familiar Olympics interlocking rings logo, a title in the top bar that says "Olympics," and a file menu with entries for "New," "Open," and "Save." Initially, only New and Open will be active. Save will be grayed out and inactive until a new Olympics has been created using the New submenu (see below) or an existing one has been loaded using Open.

If the user chooses Open, your program will present the user with a file picker and allow him or her to select an appropriate file for loading via object input. If the user chooses Save once that option becomes available, your program will present the user with a file picker and allow him or her to designate an appropriate file for saving via object output.

Under "New" there will be sub-options for "Olympics," "Event," "Team," "Athlete," and "Participation Record." Until a new Olympics has been created or a previously saved Olympics has been loaded from a file, all of these options will be grayed out and non-functional with the exception of "Olympics." Once an Olympics has been created or loaded from a file, the options for New→Event and New→Team will become active. In addition, the name of the Olympics will appear on the Home View above the Olympic rings logo, the year of the Olympics will be superimposed in a large font across the Olympics rings logo, and the host city location will appear below the Olympic rings logo. Once a team has been created or loaded from a file, the option for New→Athlete will become active. Once an athlete and an event have both been created and/or loaded from a file, the option for New→Participation Record will become active.

New Olympics View:

If the user chooses New→Olympics, your program will check to see if there is unsaved model data for an existing Olympics. If there is, your program will provide the user with a dialog asking if the unsaved data should be saved or discarded or if the user wishes to cancel the creation of the new Olympics. If

the user chooses cancel or closes the dialog window without explicitly choosing to save, discard, or cancel, your program will return to the Home View without saving or discarding the data in the model. If the user chooses the save option, your program will allow the user to select a file using a **JFileChooser** dialog and save the model data to that file.

Once the issue of unsaved data (if any) is resolved, the *New Olympics View* will appear. This will be a window that asks the user for the name, year, and host city location of the new Olympics. In addition, it will have buttons marked "Create" and "Cancel." If the user clicks "Cancel" or closes the view without explicitly choosing "Create" or "Cancel," your program will return to the Home View without altering the model. If the user chooses "Create," your program will verify that all fields (name, year, and host city location) contain data. If they do not, the user will be notified that all fields must be filled in to create a new Olympics. If they do, the existing data in the model will be discarded and a replaced with the new data from the New Olympics View.

New Event View:

If the user chooses New→Event, the *New Event View* will appear. This will be a window that asks the user for the name of the event and the team size for this event (one for an individual event, etc.). In addition, it will have buttons marked "Create" and "Cancel." If the user clicks "Cancel" or closes the view without explicitly choosing "Create" or "Cancel," your program will return to the Home View without altering the model. If the user chooses "Create," your program will verify that both fields (name and team size) contain data. If they do not, the user will be notified that both fields must be filled in to create a new event. If they do, your program will check to see if an event with that name already exists for this Olympics. If an event with that name already exists for this Olypics, the user will be notified of that fact via a dialog window and told that for that reason the new event cannot be created; the user will then be returned to the Home View and the model will not change. If an event with the given name does not already exist for this Olympics, a new event will be added to the current Olympics based on the data from the New Event View.

New Team View:

If the user chooses New→Team, the *New Team View* will appear. This will be a window that asks the user for the name of the country that the team represents. In addition, it will have buttons marked "Create" and "Cancel." If the user clicks "Cancel" or closes the view without explicitly choosing "Create" or "Cancel," your program will return to the Home View without altering the model. If the user chooses "Create," your program will verify that the field (for country name) contains data. If it does not, the user will be notified that a country name must be given to create a new team. If it does, your program will check to see if a team with that country name already exists for this Olympics. If a team with that country name already exists for this Olympics, the user will be notified of that fact via a dialog window and told that for that reason the new team cannot be created; the user will then be returned to the Home View and the model will not change. If a team with the given country name does not already exist for this Olympics, a new team will be added to the current Olympics based on the data from the New Team View.

New Athlete View:

If the user chooses New→Athlete, the *New Athlete View* will appear. This will be a window that asks the user for the team with which the athlete is associated and the name of the athlete. The user will select the team from the list of teams already created or loaded from a file. In addition, it will have buttons marked "Create" and "Cancel." If the user clicks "Cancel" or closes the view without explicitly

choosing “Create” or “Cancel,” your program will return to the Home View without altering the model. If the user chooses “Create,” your program will verify that both fields (for team and athlete name) contain data. If they do not, the user will be notified that a team must be selected from those teams in the current Olympics and that an athlete name must be given to create a new athlete. If they do, your program will check to see if an athlete with that name already exists for this team. If an athlete with that name already exists for this team, the user will be notified of that fact via a dialog window and told that for that reason the new athlete cannot be created; the user will then be returned to the Home View and the model will not change. If an athlete with the given name does not already exist for this country, a new athlete will be added to the team based on the data from the New Athlete View.

New Participation Record View:

If the user chooses New→Participation Record, the *New Participation Record View* will appear. This will be a window that allows the user to select an event, one or more athletes (depending on the event selected), an outcome (winning gold, silver, bronze, or no medal), and then create the new participation record or cancel. All of this functionality will be visible on the New Participation Record View when it becomes visible but only event selection and “Cancel” will initially be active. The user will be able to select the event from the list of events already created or loaded from a file. Once the event is selected, the New Participation Record View will tell the user how many athletes need to be selected for this event (based on the team size for the event). Then the user will be presented with a list of teams that contain at least the number of available athletes required for this event. (Here “available” means that the athletes in question have not already been associated with an event of this type. For example, if an event requires four athletes and a particular national team has seven athletes on its roster, then that national team could only have one participation record for this event, because they would need four athletes for the first participation record. Once that record had been created, the team in question would only have three athletes available for that event.) If there are no such teams, the user will be notified of this fact and be returned to the Home View without any updates to the model. If the user selects a team with sufficient available athletes from the presented list, he or she will next be presented with a list of available athletes for that team and will be able to select from that list an appropriate number of athletes. Once the user selects the appropriate number of athletes, the outcome list will become active. Once the users selects an outcome, the “Create” button will become active. If the user clicks “Create,” your program will create the corresponding participation record and return to the Home View. If at any time the user clicks “Cancel” or closes the view without explicitly choosing “Create” or “Cancel,” your program will return to the Home View without altering the model.

Controller

Besides at least one model and at least one view, every GUI-based program using the MVC design pattern needs to have at least one controller. The controller is responsible for connecting the model(s) to the view(s) so that appropriate actions are taken in response to user gestures and that appropriate representations of data are presented to users.

For this project, the appropriate actions for user gestures and appropriate representations of data are described above, under the individual views. The controller that you create for this project, then, will need to ensure that the actions are taken and views updated as described above. Call this controller **OlympicsController**.

How to Complete this Project:

1. Create figures, on engineering paper or using drawing software, to show approximately what each view will contain. These figures do not need to exactly match the appearance of the final windows but should contain all major components and show their basic layout.

2. Revise your UML design from the lab session. Be sure to clearly write your name and the names of your group members and "Project #4" on the cover sheet. **Remember to not include any personally identifying information on your project other than on your cover sheet.** Make sure to keep a copy of your UML when you turn it in.

3. Create the classes and methods specified in your design, but do not put code in the methods. Add the required documentation to your classes and methods as specified in the documentation requirements posted on the class website. This is called "stubbing" your classes and methods.

4. Run your stubbed Java files through Javadoc as described in the Lab #2 slides. This will create a set of HTML files in a directory named "docs" under your project directory.

5. Submit your **view figures**, UML design, stub code, and Javadoc as your initial design. (See below for due dates and requirements regarding submission of paper and electronic copies of project components.)

6. Implement the design you have developed by coding each method you have defined as well as any others you have left out of your design. As you do this, make sure to modify and annotate the changes to your design on your UML and properly document all new code. A good approach to the implementation of your project is to follow the project's milestones in the order they have been supplied.

7. Test your program and fix any bugs.

8. Once you have completed the project and are ready to submit it for grading, create a new set of Javadoc files using Eclipse and inspect them to make sure your detailed design is properly documented in your source code. (Actually, it is a good practice to keep your Javadocs up to date as you develop your software so that they can be of use to you and your team members and to the instructor and/or TA if you come to office hours for help.)

9. Submit all parts of your completed project. (See below for due dates and requirements regarding submission of paper and electronic copies of project components.)

Extra Credit Features:

You may extend this project with more search features for an extra 5 points of credit. For example, you could think of ways to present the user with helpful information about the program, such as a context-dependent help system.

To receive the full five points of extra credit, your extended features must be novel (unique) and must involve effort in the design of the extra features and their integration into the project and the actual coding of the features. Also, you must indicate on your final UML design which portions of the design support the extra feature(s); and you must include a write-up of the feature(s) in your milestones.txt file. The write-up must indicate what each feature is, how it works, how it is unique, and the write-up must cite any outside resources used.

Due Dates and Notes:

An electronic copy of your revised design including **view figures** (optional in electronic version), UML (optional in electronic version), stub code, and detailed Javadoc are due on **Wednesday, April 7th**. Submit the project archive following the steps given in the submission instructions **by midnight**. Submit your **view figures on *engineering paper* or hardcopies made using drawing software**, revised UML design on *engineering paper* or a hardcopy made using UML layout software, and your cover page at the **beginning of lab on Thursday, April 8th**.

An electronic copy of the final version of the project is due on **Thursday, April 22nd**. Submit the project archive following the steps given in the submission instructions **by midnight**. Submit your **final view figures on *engineering paper* or hardcopies made using drawing software**, final UML design on *engineering paper* or a hardcopy made using UML layout software, a hardcopy of the cover page for your project, and a hardcopy of the milestones.txt file at the **beginning of lecture on Friday, April 23rd**.

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.

As noted in the syllabus, you are required to work on this programming assignment in a group of at least two people. It is your responsibility to find other group members and work with them. The group should turn in only one (1) hard copy and one (1) electronic copy of the assignment. Both the electronic and hard copies should contain the names and student ID numbers of all group members on the cover sheet. If your group composition changes during the course of working on this assignment (for example, a group of five splits into a group of two and a separate group of three), this must be clearly indicated in your cover sheet, including the names and student ID numbers of everyone involved and details of when the change occurred and who accomplished what before and after the change.

Each group member is required to contribute equally to each project, as far as is possible. You must thoroughly document which group members were involved in each part of the project. For example, if you have three functions in your program and one function was written by group member one, the second was written by group member two, and the third was written jointly and equally by group members three and four, your cover sheet must clearly indicate this division of labor. Giving improper credit to group members is academic misconduct and grounds for penalties in accordance with school policies.