Assignment 1 – 2011

Create public class *FilmFestivalException*, which represents an unchecked exception (extends the *RuntimeException* class) and has:

- Public constructor that receives, as its input parameter, an error message and passes this parameter to the constructor of the parent class.

Create public class *FilmFestival* with the following elements:

- Private attribute *name* with the initial value null.
- Private attribute *location* with the initial value null.
- Private attribute *numberOfVisitors* representing the number of people who visited the film festival.
- Get and set methods for all the attributes. The *name* and the *location* attributes cannot be null nor empty string, while the number of participants has to be either zero or a positive number. In case of invalid parameter value, a *FilmFestivalException*, with an appropriate error message, should be thrown.
- Redefined *toString* method (of the Object class) so that it returns a piece of text with all the data about the film festival. If the number of visitors is equal to zero, the method should return a message (string) with the festival name and location only.
- Redefined *equals* method (of the Object class). First, it should be checked if the method's input is an object of the *FilmFestival* class; if it is not, a *FilmFestivalException*, with an appropriate error message, should be thrown. If the input is a *FilmFestival* object, the method returns true, if name and location of the festival are equal to the name and the location of the input *FilmFestival* object; if these are not equal, the method returns false.

Create public class *FilmFestivalSeason* with:

- Private attribute *festivals* representing a list of objects of the *FilmFestival* class.
- Public constructor without input parameters; the constructor initializes the *festivals* list.
- Public method that writes to the text file "report.txt" data about each film festival (from the *festivals* list) with more than 100 visitors. Data about each film festival should be written in a separate row.
- Public method that reads from the keyboard the data about one film festival, and adds the new festival to the list. The new festival should be added to the list only if the same festival is not already in the list. In case of an error occurring while reading the data, that is, if an exception is thrown, the exception should be caught and its message should be printed in the console.
- Public method that receives, as its input parameter, a list of the *FilmFestival* objects. This is a list of festivals from the previous season. The method writes to the "popular_festivals.txt" text file data about those festivals that were held both this and the previous season, and in this season had more visitors than in the previous season. Note that the number and order of festivals in the two lists may differ.

Assignment 5 – 2011

Create public class *CityException* that represents an unchecked exception (extends the *RuntimeException* class) and has:

- Public constructor that receives, as its input parameter, an error message and passes this parameter to the constructor of the parent class.

Create public class *City* that can be serialized, and has:

- Private attribute *name* with the initial value "unknown".
- Private attribute *population* that represents the number of citizens in the given city; its initial value is zero.
- Get and set methods for these two attributes. Invalid values for the attribute *name* include null and any String with less than two letters. The city's *population* has to be greater than zero. In case of an invalid value, a *CityException*, with an appropriate error message, should be thrown.
- Redefined *equals* method (of the Object class). The method first checks if its input is an object of the *City* class; if it is not, a *CityException*, with an appropriate error message, should be thrown. If the input is a *City* object, the method returns true, if name of the city is equal to the name of the input *City* object; if these are not equal, the method returns false.

Create visual class *CitiesGUI* that looks like the one shown on Figure 1. The title of the window should be "CitiesGUI". Set the window in such a way that its size cannot be changed by the user.

- The *CitiesGUI* class should have private attribute *cities* that represents a list of objects of the *City* class; the list should be initialized right away.

- When the Delete button is pressed, the content of both input fields should be deleted.
- When the Save button is pressed, data about all the cities in the list should be written (serialized) into 3 files: "small_cities.out", "midsize_cities.out", and "big_cities.out", depending on the cities' population. Cities with less than 100 000 citizens are considered small; midsize cities have between 100 000 and 1 million citizens; big cities are those with over 1 million citizens.
- When the Add button is pressed, the data about the name and the population of a new city should be taken from the input fields and the new city should be added to the list. The new city is added only if the list doesn't already contain the same city. The new city, if not in the list, should be added to it in such a way that the descending order based on the cities' population is preserved in the list. If the list already contains the same city or an exception is thrown while entering or transforming the data, the word "ERROR" should be written in both text fields.

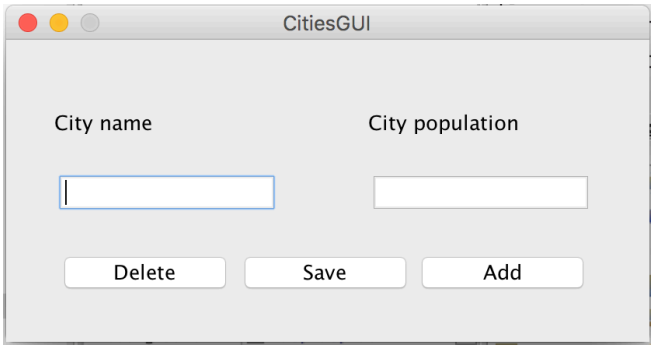

Figure 1. CityGUI

Assignment 6 – 2011

Create public class *CityException* that represents an unchecked exception (extends the *RuntimeException* class) and has:
- Public constructor that receives, as its input parameter, an error message and passes this parameter to the constructor of the parent class.

Create public class *City* that can be serialized, and has:
- Private attribute *name*.
- Private attribute *population* that represents the number of citizens in the given city.
- Get and set methods for these two attributes. Invalid value for the attribute *name* is null, while the city's *population* has to be greater than zero. In case of an invalid value, a *CityException*, with an appropriate error message, should be thrown.
- Redefined method *toString* (of the Object class) that returns a piece of text (String) with all the data about the city, in the following format: "CITY NAME: #### POPULATION: ###".

Create visual class *CitiesGUI* that looks like the one shown on Figure 2. The window title should be "List of cities", and the central part of the window should contain text editor. When a user resizes the window, the central part (with the text editor) should be enlarged/shrunk, while the other components should stay unchanged (see Figure 3).
- The *CitiesGUI* class should contain private attribute *cities* that represents a list of objects of the *City* class; the list should be initialized right away.
- When the "Exit" button is pressed, the program should be terminated.
- When the "Load" button is pressed, data about cities should be loaded from two files: "archive1.out" and "archive2.out", and the loaded cities should be used to fill in the *cities* list, but without repetition (if the same city appears in both files, it should be added to the list just once). Before loading the data from the files, the list should be cleared.
- When the "Print" button is pressed, data about three cities (from the list) with the largest population should be printed in the editor. The data about each city should be printed in a separate row. If the list is empty, message "List is empty" should be printed in the editor.

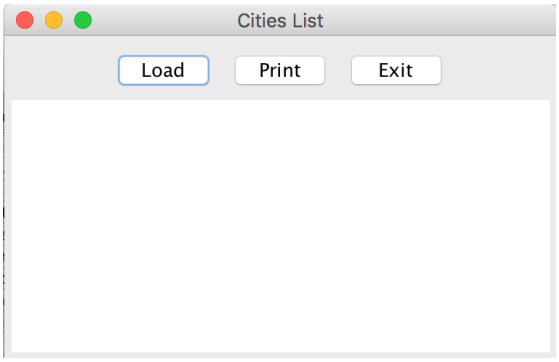

Figure 2. CitiesGUI (regular size)

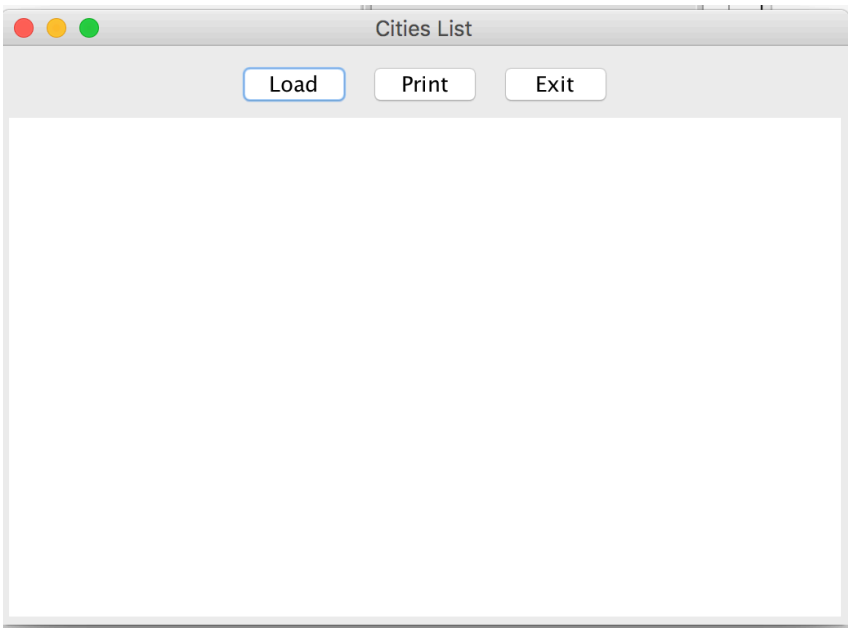

Figure 3. CitiesGUI (enlarged)

Assignment 7 – 2011

Create public class *WaterPoloClubException* that represents an unchecked exception (extends the *RuntimeException* class) and has:

- Public constructor that receives, as its input parameter, an error message and passes this parameter to the constructor of the parent class.

Create public class *WaterPoloPlayer* that has:

- Private attribute *fullName*; value of this attribute is always in the format "NAME SURNAME".
- Private attribute *position*, which represents the position of the player in the game (e.g., goalkeeper, quarterback)
- Private attribute *score* that represents the total number of points the player scored during a game.
- Get and set methods for these three attributes. Invalid values for the *fullName* and the *position* attributes include null and empty strings, while the score has to be either zero or greater than zero. In case of an invalid value, a *WaterPoloClubException*, with an appropriate error message, should be thrown.
- Redefined *equals* method (of the Object class). The method first checks if its input is an object of the *WaterPoloPlayer* class; if it is not, a *WaterPoloClubException*, with an appropriate error message, should be thrown. If the input is a *WaterPoloPlayer* object, the method returns true, if full name of the water polo player is equal to the full name of the input *WaterPoloPlayer* object; if these are not equal, the method returns false.

Create visual class *WaterPoloClubGUI* that looks like the one shown on Figure 4. The title of the GUI window should be "Water polo club". Set the window in such a way that users cannot change its size. The dropdown list should have the following items: "goalkeeper", "wing", "quarterback", "anchor", and "center".

- The *WaterPoloClubGUI* class should contain private attribute *players*, which is a list of objects of the *WaterPoloPlayer* class. The list should be initialized right away.
- When the "Delete" button is pressed, the content of all the input fields (except the dropdown list) should be deleted.
- When the "Save" button is pressed, data about all the water polo players (from the players list) should be written into two files "goalkeepers.out" and "players.out". In the first file ("goalkeepers.out"), only the data about the goalkeepers should be written, while the second file ("players.out") should store the data about all the other players from the list. In both cases, data about each player should be written in a separate row, in the following format: <full_name><tab><position><tab><score>.
- When the "Add" button is pressed, all the data about a water polo player should be collected from the input fields of the GUI, and a new instance of the *WaterPoloPlayer* should be created and added to the list. The new player should be added only if the list doesn't contain the same player. The new player, if not in the list, should be added to it in such a way that the descending order based on the players' scores is preserved in the list. If the list already contains the same player or an exception is thrown while entering or transforming the data, the word "ERROR" should be added to the title of the GUI window.

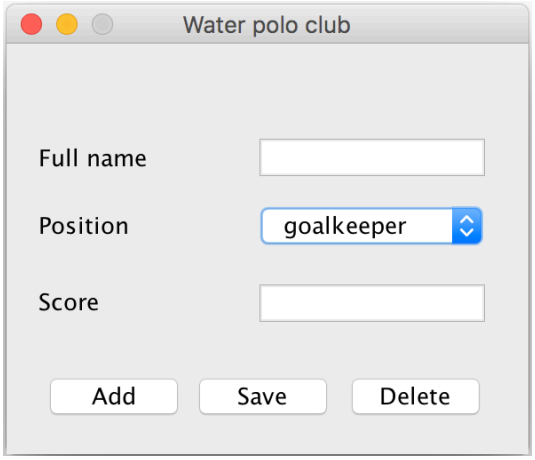

Figure 4. WaterPoloClubGUI

Assignment 4 – 2012

Create public class *HRException* that represents a checked exception (extends the *Exception* class) and has:
- Public constructor that receives, as its input parameter, an error message and passes this parameter to the constructor of the parent class.

Create public class *Employee* that can be serialized and has the following elements:
- Private attribute *fullName*.
- Private attribute *age*.
- Private attribute *yearsEmployed* that represents the number of years the given person has been employed.
- Private attribute *gender* that can take one of the following two values: 'f' for females, and 'm' for males.
- Get and set methods for these attributes. Invalid values for the *fullName* include null and strings containing more than 20 characters, while the *age* and the *yearsEmployed* attributes have to be either zero or greater than zero. The only allowed values for *gender* are 'f' and 'm'. In case of an invalid value being entered, an *HRException*, with an appropriate error message, should be thrown.
- Redefined method *toString* (of the Object class) that returns a piece of text (String) with all the data about the employee; the text should be formatted in such a way that after each employee attribute, there is one "tab" sign.
- Redefined *equals* method (of the Object class). The method returns true if the full name and the age of the employee are equal to the full name and the age of the input *Employee* object; if any these attributes are not equal, the method returns false.

Create public class *HRSystem* that has:
- Private attribute *employees*, which is a list of objects of the *Employee* class; the list should be initialized right away.

- Public method that writes to the "report.out" file data about those employees who should be given jubilee salary; the data should be written in the following format: <full_name><years_employed>. The employees who are granted the jubilee salary are those who have been employed for at least 10 years.
- Public method that serializes to the "near_pension.out" file data about those employees who are ready for pension. Those are male employees with at least 65 years of age or at least 40 years of employment, as well as female employees with at least 60 years of age or at least 35 years of employment. In addition, the method should print to the screen the total number of employees who are ready for pension.
- Public method that reads from the keyboard data about one employee and adds him/her to the list. In case of an error (exception) occurring while reading the value for any of the employee attribute, an error message should be printed to the screen, and another attempt should be made at reading the value of that attribute.

Assignment 5 – 2012

Create public class *DemographyException* that represents an unchecked exception (extends the *RuntimeException* class) and has:
- Public constructor that receives, as its input parameter, an error message and passes this parameter to the constructor of the parent class.

Create public class *Region* that can be serialized and has the following elements:
- Private attribute *name*.
- Private attribute *birthRate* that represents the total number of children born since the latest census.
- Private attribute *deathRate* that represents the total number of people who died since the latest census.
- Private attribute *migrationBalance* that represents the change in the population size due to migration (it is an integer value)
- Get and set methods for these attributes. Invalid values for the *name* attribute include null and strings containing less than 2 letters, while the *birthRate* and the *deathRate* attributes have to be greater than zero. In case of an invalid value being entered, a *DemographyException*, with an appropriate error message, should be thrown.
- Redefined toString method (of the Object class). The method returns a piece of text (String) with all the data about the region. The text to be returned should also contain information about the change in the population size, which is computed as follows: change = birthrate – death rate + migration balance.

Create visual class *RegionsGUI* that looks like the one shown on Figure 5. The title of the GUI window should be "Regional demographic data", and the central part of the window should contain text editor. When a user resizes the window, the central part (with the text editor) should be enlarged/shrunk, while the other components should stay unchanged.
- The *RegionGUI* class should contain private attribute *regions*, which is a list of objects of the *Region* class; the list should be initialized right away.
- When the "Delete" button is pressed, the content of all the text fields and the text area should be deleted.
- When the "Save" button is pressed, all the elements of the *regions* list should be written (serialized) into one of these two files: "growing_regions.out" and "dying_regions.out", depending on the change in the population size (this change is computed using the formula: change = birthrate – death rate + migration balance). Those regions where the change is positive should be written into the first file, while the others should be written to the second file.
- When the "Add" button is pressed, all the data about a region should be collected from the input fields of the GUI, and a new instance of the *Region* should be created and added to the list. The new region should be added only if the list doesn't contain the same region. The new region, if not in the list, should be added to it in such a way that the descending order based on the regions' birthrate is preserved in the list. If the list already contains the same region or an exception is thrown while entering or transforming the data, the word "ERROR" should be printed in the text editor.

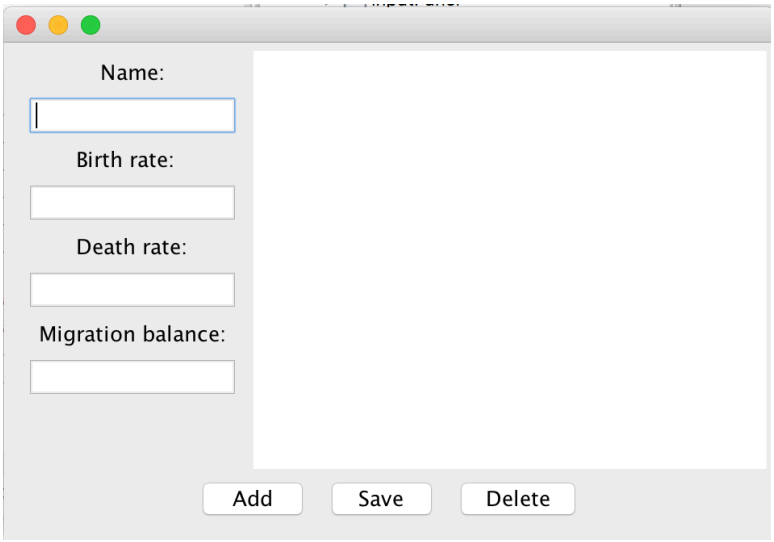

Figure 5. RegionsGUI

Assignment 3 – 2012

Create public class *StatisticsException* that represents a checked exception (extends the *Exception* class) and has:

- Public constructor that receives, as its input parameter, an error message and passes this parameter to the constructor of the parent class.

Create public class *Household* that can be serialized and has:

- Private attribute *location* representing the name of the place where the household is located
- Private attribute *grownupsNum* that represents the number of grownups in the household
- Private attribute *childrenNum* that represents the number of children in the household.
- Private attribute *monthlyIncome* that represents the amount of the total monthly income of the household in dinars.
- Get and set methods for these attributes. Invalid values for the *location* attribute include null and strings with less than 5 or more than 13 characters; the other three attributes must be either zero or greater than zero. In case of an invalid value being entered, a *StatisticsException*, with an appropriate error message, should be thrown.
- Redefined method *toString* (of the Object class) that returns a piece of text (String) with all the data about the household; the text should be formatted in such a way that after each household attribute, there is one “tab” sign.
- Redefined *equals* method (of the Object class). The method returns true if values of all the attributes of the household are equal to the values of the corresponding attributes of the input *Household* object; if any these attributes are not equal, the method returns false.

Create class *HouseholdsStatistics* with the following elements:

- Private attribute *households*, which represents a list of objects of the *Household* class.
- Public constructor with no input parameters, which initializes the *households* list.
- Public method that writes to the data file “households_without_income.out” data about the households with zero monthly income; the data about each such household should be written in the following format: <location><grownups_number><children_number>. If the *households* list is empty, an exception of the type *StatisticsException* should be thrown.
- Public method that creates a report based on the data stored in the *households* list and writes this report to the file “report.txt”. The report should contain the total number of households, the average monthly income per household, the average number of grownups per household, the average number of children per household, as well as the average monthly income per household member (including both grownups and children).
- Public method that reads data about several households and adds them to the list. The number of households to be added is read (entered by the user) at the first input. In case of an error (exception) occurring while reading the data for a household, an error message should be printed to the screen, and another attempt should be made at reading the data for that household.

Assignment 8 – 2012

Create public class *EmployeeException* that represents an unchecked exception (extends the *RuntimeException* class) and has:

- Public constructor that receives, as its input parameter, an error message and passes this parameter to the constructor of the parent class.

Create public class *Employee* that can be serialized and has:

- Private attribute *fullName*.
- Private attribute *age*.
- Private attribute *yearsEmployed* that represents the number of years the employee has been employed.
- Private attribute *female* with value TRUE if the employee is female, and FALSE if the employee is male.
- Get and set methods for these attributes. Invalid values for the *fullName* include null and strings with more than 20 characters, while the *age* and the *yearsEmployed* attributes have to be either zero or greater than zero. In case of an invalid value being entered, an *EmployeeeException*, with an appropriate error message, should be thrown.
- Redefined method *toString* (of the Object class) that returns a piece of text (String) with all the data about the employee; the text should be formatted in such a way that after each employee attribute, there is one "tab" sign.

Create visual class *EmployeesGUI* that looks like the one shown on Figure 6a. The title of the GUI window should be "Employees", and the central part of the window should contain text editor. When a user resizes the window, the central part (with the text editor) should be enlarged/shrunk, while the other components should stay unchanged (Figure 6b). The dropdown list should have two items: "male" and "female".

- The *EmployeeGUI* class should contain private attribute *employees*, which is a list of objects of the *Employee* class; the list should be initialized right away.
- When the "Delete" button is pressed, the content of all the text fields and the text area should be deleted.
- When the "Load" button is pressed, data about employees should be read (deserialized) from a file; the name of the file (to be used for deserialization) should be read from the text editor; once deserialized, the objects should be added to the *employees* list. After deserialization, the content of the *employees* list should be written in the text editor.
- When the "Add" button is pressed, all the data about an employee should be collected from the input fields of the GUI, and a new instance of the *Employee* class should be created and added to the list. The new employee should be added only if the list doesn't contain the same employee. If an exception is thrown while entering or transforming the data, the word "ERROR" should be written into the appropriate input field (for instance, if -1 is entered as the value for the years of employment, "ERROR" should be written into the text field for entering the years of employment).

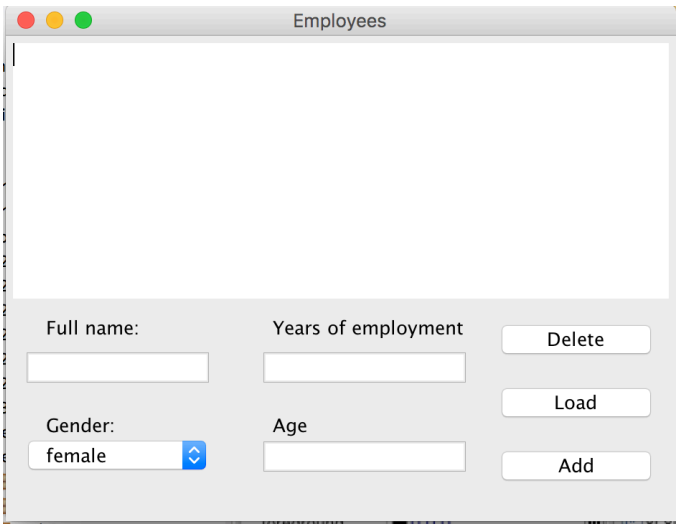

Figure 6a. EmployeesGUI (initial size)

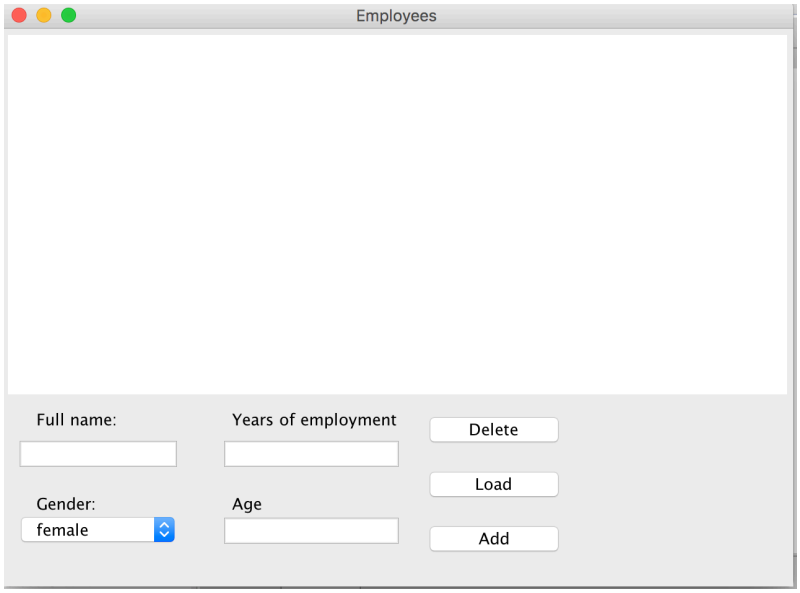

Figure 6b. EmployeesGUI (after being enlarged)