**CIS 163 Project 2 – A Game Store Rental System**

**THIS IS A GROUP (2 - 3 members) PROJECT.**

**Due Date**

* See schedule at the end of the syllabus

**Why are we doing this project.**

**The instructor will fill in the details; here are some bullet points**

* Whether games are being rented is irrelevant, most companies have an inventory of items to track.
* In most projects, partial source code will have been developed, so working with old code is important.
* In most projects today that are small (mid) size, client involvement is a must. In some projects, involvement could be weekly, while other projects could be less frequent. For us in a classroom setting, I expect questions from groups each class period, so I know (as the client) work is being done from the start of the project.
* There are several major requirements that are difficult (save/load text, using streams, and figuring out how this current program functions). I am happy to help any group that meets with me or asks questions in class.
* **This project is challenging**,

**Before Starting the Project**

* Review Chapters 8 - 10 and Chapters 12, 13, 15, 18 of the CIS163 book
* Read this entire project description before starting if you have any question please ask the instructor

**Learning Objectives**

After completing this project, you should be able to:

* Use inheritance and polymorphism
* Use advanced Swing components like JTable and AbstractTableModel
* Save and restore objects using the **Serialization/text** files
* Use Collections.sort
* Using simple Date and GregorianCalendar classes
* Using streams and Lamda functions
* Using anonymous classes

**Program description:** Your assignment is to create a program that helps camping owners manage their inventory. You should be able to rent games and Consoles sites from your program. A full description follows.

**A completed program must have the following functionality:**

* Save, load the rental database with serialized files using JFileChooser
* Save, load the rental database with text files using JFileChooser
* Reserve a Games or Consoles site with a checkin date, checkout date, name of camper
* Complete error checking
* **And much more! DETAILS BELOW**

**Important: All the following Steps were provided to you in the sample code. In other words, the following is just a description to what has been provided to help you better understand the code.**

**Step 0:** I have provided a functioning program that will get you started on this project. The intent of the code is to help you understand the different techniques that will be used in your final project. This sample program has many issues, such as no error checking, incomplete results and most importantly does not implement most of the functionality that is required in your project. **There is much that must be changed on the sample code; it should get your team thinking about how best to proceed.**

**Step 1: Import the project starter files**

**Step 2: Rental (base class) is implemented using the following:**

**(This has been done for you, i.e., in the staring code)**

public abstract class Rental implements Serializable {  
 */\*\* What is the purpose of this variable (search google) \*/* private static final long *serialVersionUID* = 1L;  
  
 */\*\* The Name of person that is reserving the Rental\*/* protected String nameOfRenter;  
  
 */\*\* The date the Rental was rented on \*/* protected GregorianCalendar rentedOn;  
  
 */\*\* The date the Rental was dueBack on \*/* protected GregorianCalendar dueBack;  
  
 */\*\* The actual date the Rental was returned on \*/* protected GregorianCalendar actualDateReturned;

*/\*\* and so on \*/*

*// add constructor*

*// add getter, setter methods*

**Step 3a: Game is a derived class by extending Rental and using the following:**

**public** **class** Game **extends** Rental {

/\*\* represents the name of the game \*/

**private** String nameGame;

/\*\*

\* Represents the console the person rented to play the game,

\* null if no console was rented.

\*/

**private** ConsoleTypes console;

*// add constructor, add getter, setter methods*

*// You will need to override the cost function here*

**Step 3b: Console is a derived class by extending Rental and using the following:**

*public class Console extends Rental {*

*/\*\* Represents the type of Console, see enum type \*/*

*private ConsoleTypes consoleType;*

*// add constructor, add getter, setter methods*

*// You will need to override the cost function here*

**Step 4: GUIRentalStore is implemented using the following:**

public class GUIRentalStore extends JFrame implements ActionListener {

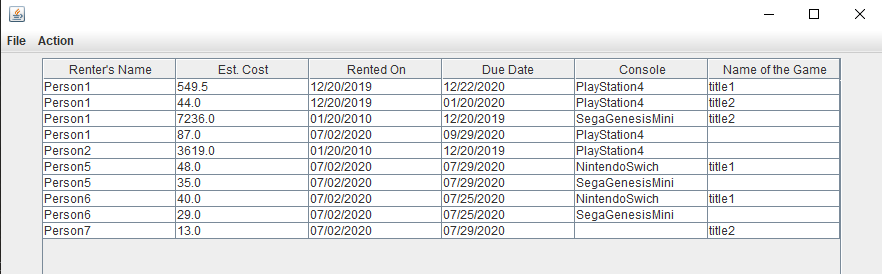
*// declare GUI components (menu items, buttons, etc.) needed*

*// constructor method that prepares the GUI*

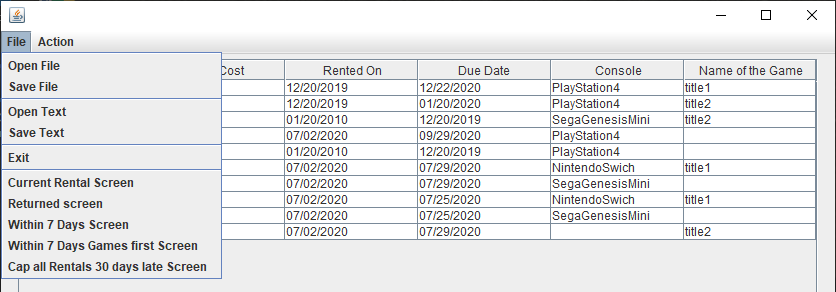
*// event handlers and other methods needed to build the GUI*

* The GUIRentalStore class is the class that displays the GUI to the user, allows the user to rent Consoles or Games, and allows the user to return the items too. In addition, the GUI allows the users to save and load the database using serialized **and text files**. The GUIRentalStore must handle the following operations shown below. The first screen shot shows the main GUI screen:

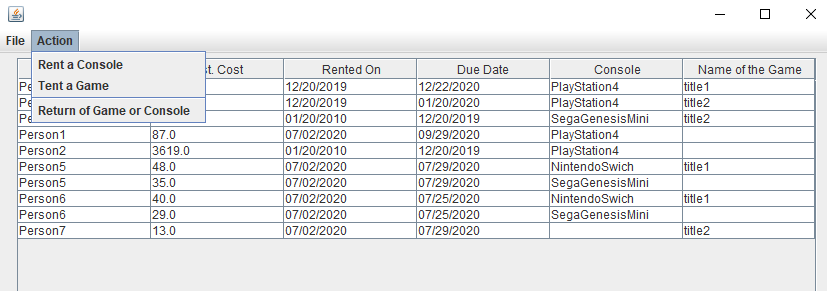
Example1



Example 2



Example 3



**Step 5: ListModel is implemented using the following**

**(THIS IS THE MOST IMPORTANT CLASS TO UNDERSTAND):**

This class is used for managing the Rentals (Games and Consoles) into an ArrayList<Rental>. (Note: Rental is the base class. Review Chapter 9 of your book for more information.) The functionality of this class is similar in concept to code presented in Chapter 9, specifically, the staffList array. The main difference is that this class must handle all the operations from the GUI class. That is, renting Games or Consoles, returning Games or Consoles, save and load, etc.

Note: The following code is just a start; to fully understand how to create the ListModel see the class notes. Examples of a ListModel class will be presented in class.

public class ListModel extends AbstractTableModel {  
  
 */\*\* holds all the rentals \*/* private ArrayList<Rental> listOfRentals;  
   
 */\*\* holds only the rentals that are to be displayed \*/* private ArrayList<Rental> fileredListRentals;  
  
 */\*\* current screen being displayed \*/* private ScreenDisplay display = ScreenDisplay.*CurrentRentalStatus*;

@Override  
public int getColumnCount() {

@Override  
public int getRowCount() {

@Override  
public Object getValueAt(int row, int col) {

@Override  
public String getColumnName(int col) {

*// add methods to add, delete, and update.*

*// add methods to load/save accounts from/to a binary/text file*

*// add other methods as needed*

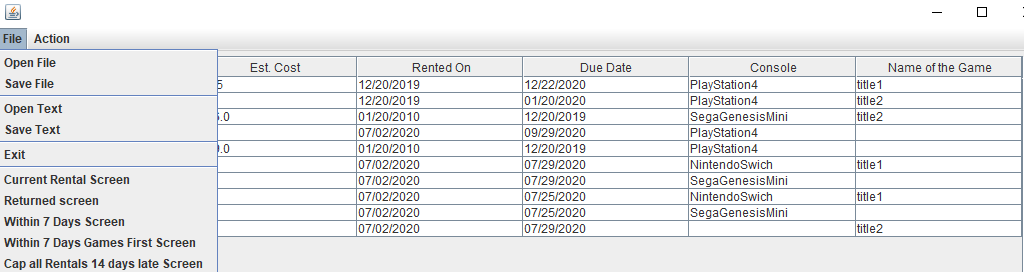
*Again, the above is just a starting point, please run the code provided to fully understand how it functions.*

**This section starts the new functionality your group must implement.**

**Task 1: To test your understanding of how the code functions:**

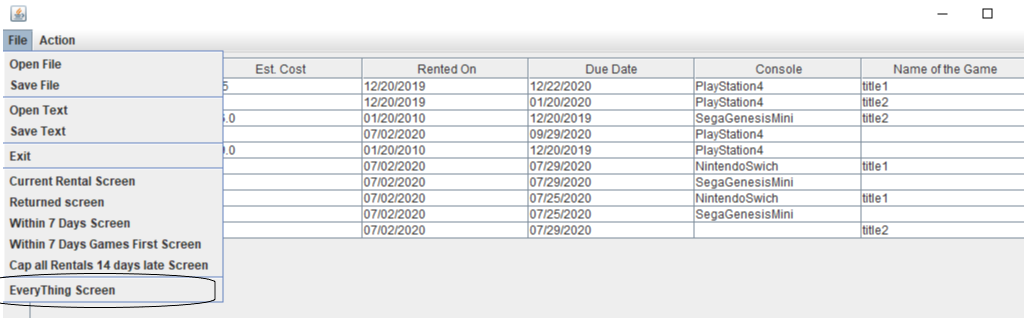
This Task is not simple if you do not understand how the program functions. So, after you have understood this code, attempt to add on a new Screen called “EveryThingScreen”. This screen will show all records within the inventory system. Make sure the Rentals are displayed in alphabetical order of the renter’s name

**Current File menu looks like:**

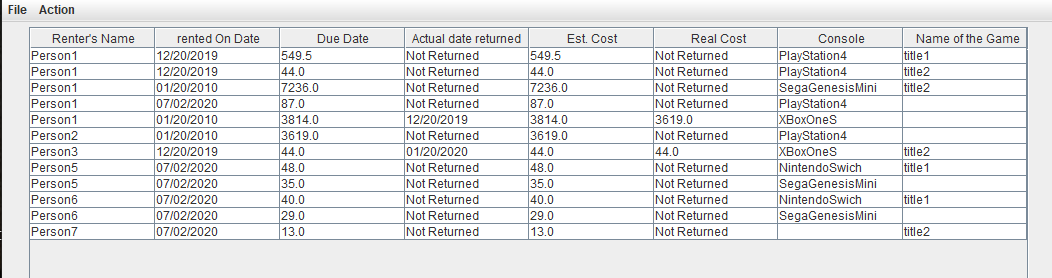


These following pictures show the new File item (EveryThing Screen) and the result output.

**New File menu looks like.**

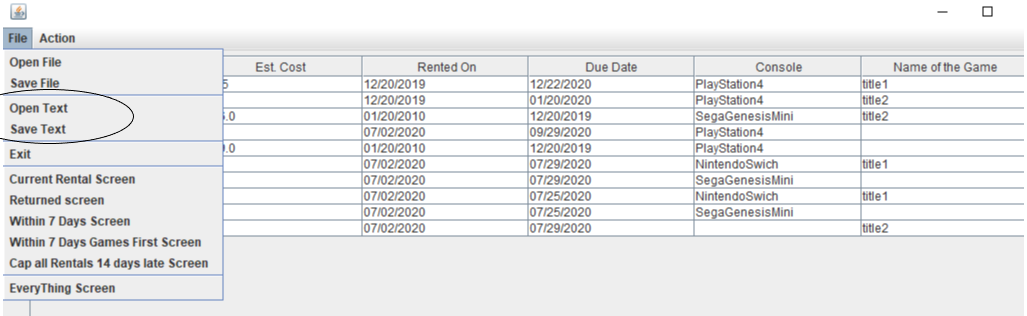
****

**What the “EveryThing Screen” should look like:**

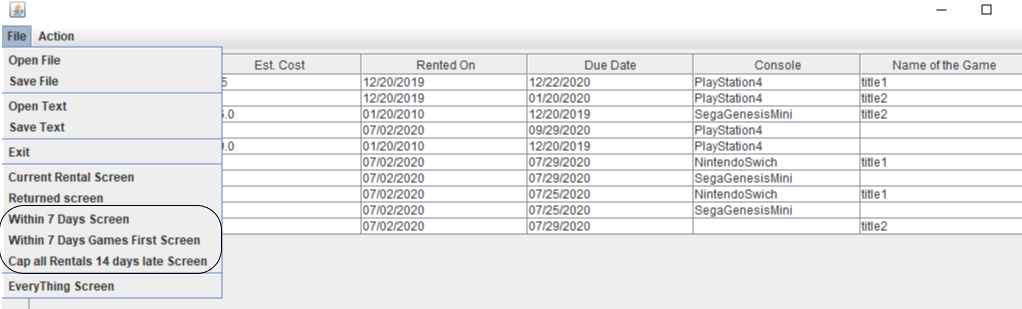


**Task 2: Add on the loading the database using a text file.**

**Your task** is to complete the loadFromText method, given the save method that is already provided to you. This is a difficult task if you do not understand the save method. **The save method cannot be modified.**

****

**Task 3 Add on 3 different screens.**

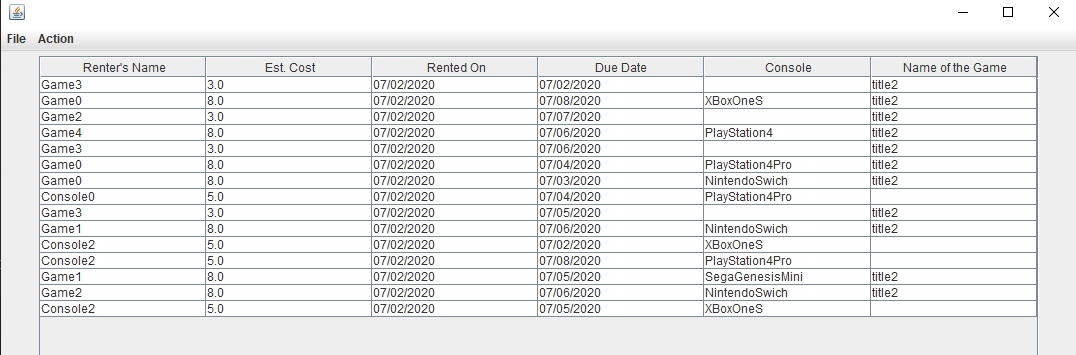
****

**Your task A:** Complete the “*Within 7 Days screen*”; to find this screen, Click on File, then click on “Within 7 Days Screen”. Complete the code found in the private void updateScreen() method. Look for // Your code goes here. The requirements for this Task are:

* Show all the rentals that are not yet returned and have less than or equal to 7 days between the rented on date and the due date
* You must use streams, lambda functions, and the Collections class to perform this task. **Note: no loops are allowed here**

Your current screen does not work correctly and must be fixed to adhere to the above requirements.

The final screen should look like this. (Note: this screen was generated by changing while (count < 0) to while (count < 30) in the public void createList() method.)

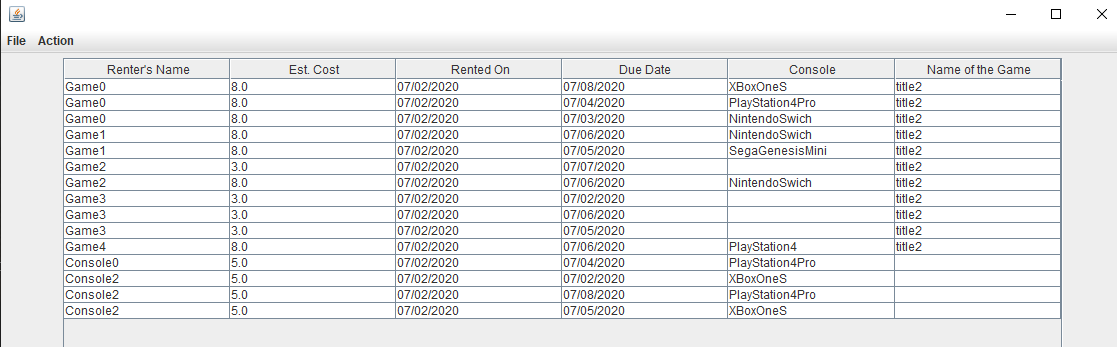


**Your task B:** Complete the “*Within 7 Days Games First screen”*; to find this screen, Click on File, then click on “Within 7 Days Games First Screen”. Complete the code found in the private void updateScreen() method. Look for // Your code goes here. The requirements for this Task are:

* Show all the rentals that are not yet returned and have less than or equal to 7 days between the rented on date and the due date
* Games are displayed first with Consoles second
* Within the Games or Consoles section of the list, all Renter’s names are in alphabetical order
* You must use streams, lambda functions, and the Collections class to perform this task. **Note: no loops are allowed here**

Your current screen does not work correctly and must be fixed to adhere to the above requirements.

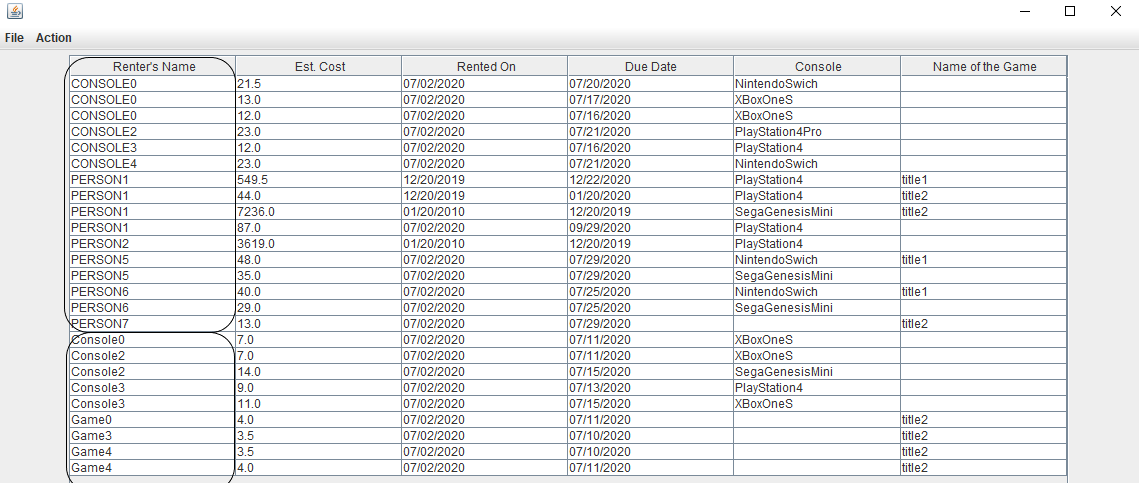
The final screen should look like this. (Note: this screen was generated by changing while (count < 0) to while (count < 30) in the public void createList() method.)



**Your task C:** Complete the “*Cap all Rentals 30 days late screen*”; to find this screen, Click on File, then click on “Cap all Rentals 30 days late screen”. Complete the code found in the private void updateScreen() method. Look for // Your code goes here. The requirements for this Task are:

* Show all the rentals that are not yet returned and have greater than 7 days between the rented on date and the due date
* Rentals that have greater than or equal to 14 days between the rented on date and the due date are capitalized (see screen below)
* Renter’s names must be in alphabetical order
* You must use streams, lambda functions, and the Collections class to perform this task. **Note: no loops are allowed here**

**FINALLY**: As you switch out of this screen to other screens, make sure you uncapitalize the Renters’ names. You can use a standard format to your screens, such as capitalizing the first letter. See your instructor if you have question regarding this task.



**Be sure to “uncapitalize” the names in the other screens.**

**Task 4: Your Task** is to select one of the following to add/implement:

* Comment out (but do not delete) the capitalization/uncapitalization code from Task 3 and find a different place in the code to implement this capitalization/uncapitalization feature such that the Rental objects themselves are unchanged. (Hint: This kind of functionality is focusing just on the *display* of the renter’s name, which means that you will need to change how the renter’s name is rendered in the GUI)
* Add a screen called “Late Rentals” that displays Rentals based on the following criteria:
  + Show all Rentals that are still rented out (based on today/current date) and are past their due date (i.e., Rentals that are currently late)
  + The table includes an additional column for “Number of Days Late” that appears between the “Due Date” column and the “Console” column
  + Games are displayed first with Consoles second
  + Within the Games or Consoles section of the list, Rentals are ordered by the number of days late it is from highest (i.e., the most days late) to lowest (i.e., the fewest days late)
  + If 2 Games or 2 Consoles are the same number of days late, then the Renter’s name is used as a tie breaker
  + You must use streams, lambda functions, and the Collections class to perform this task. **Note: no loops are allowed here**
  + You must ensure proper capitalization/uncapitalization of the Renter’s names when switching between screens (as in Step 3)
* Implement a new/different type of Rental object (i.e., something different from a Game or Console) that would make sense for a game store to offer. Make sure that this new type of Rental can be rented and returned, and that it can appear alongside Games and Consoles in the GUI. You may add additional columns of information in the GUI to help you display information for this new type of Rental, but make sure any new columns you add are to the right of the Name of Game column (i.e., all the way to the right in the GUI)
* Add on your own new and cool idea! You must get instructor approval before deciding on or implementing your own idea

**Task 5: Tour Task, TOTALLY error checked THE WHOLE program.** For example: CheckOut Date was before checkIn date; improper date such as: “abc/abc/abc”; etc.

**Your Task, Comment all of your code,** but use the Java Style Guide on ListModel. That class will be graded.

**Important, I will have test data to use during project demonstrations. The data found in the createList() method in ListModel.**

--------------------------------------------------------YOU’RE DONE ☺ ------------------------------------------------------

**CIS 163 – Computer Science II**

**Project 2:**

|  |  |
| --- | --- |
| Student Name |  |
| Date Submitted, Days Late, Late Penalty |  |

|  |  |  |
| --- | --- | --- |
| **Graded Item** | **Pts** | **Points Secured / Comments** |
| Javadoc Comments and Coding Style/Technique  (<https://www.cis.gvsu.edu/java-coding-style-guide/>)   * Code Indentation (campSite format source code in IDE) * Naming Conventions (see Java style guide) * Proper access modifiers for fields and methods * Use of helper (private) methods * Using good variable names * Header/class comments * Every method uses @param and @return (1 sentence after) * Every method uses a /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* separator * Overall layout, readability, **No text wrap** * Using /\*\* … / for each Instance variable * Has many inner “inner” comments | 10 |  |
| * Task 1, new screen (EveryThing) * Task 2, load method | 10  12 |  |
| * Task 3a, Within 7 day * Task 3b, Within 7 days Games first * Task 3c, Cap all Rentals … * Task 4, Your new idea * Task 5, Error checking | 12  12  12  12  12 |  |
| **Cleaning up the existing code that you used in your project. In other words, have a good design, no wasted lines of code, No extra code. etc.**  **MISC stuff** | 10 |  |
| **Total** | **100** |  |

**Group work grading**

Working in a group is an important part of this project, and most likely, working in a group will be a part of most computing projects you will work on in the future. Regarding Project 2, no specific points as it relates to group work will be allocated on the rubric. Instead, the instructor reserves the right to increase or decrease points to anyone's score for Project 2 for groups that are dysfunctional. Characteristics of a dysfunctional group includes (but not limited to): unequal workloads, poor communication, or not prepared for demonstrations.

Every student is required to understand how Project 2 functions during demonstration day. I will quiz students on how their project functions.

**Comments**