IB COMPUTER SCIENCE
INTERNAL ASSESMENT: PROGRAM DOSSIER
VIDEO GAME DATABASE

SEAN HOFFMAN
PERIOD 2
03/23/10

Table of Contents

A1: Problem Analysis

Since the creation of the first interactive electronic video game in 1947 by Thomas T. Goldsmith Jr. and Estle Ray Mann, video games have become increasingly more advanced. Today, video games are mass marketed by the millions creating a new category of collectable media. With such a high demand for the new form of entertainment, video game companies collectively manufacture over 200 different game titles every year. Over 100 video games platforms in existence today, some of these platforms have exclusive video games while other video games exist across multiple platforms. With such a large output of video games each year it will be difficult for the consumers to keep track of each and every different video game they purchase. In most cases a consumer must resort to using a pencil and paper method in order to keep track of this type of information, which is subject to human error.

The objective of this project is to create a video game database that is able to collectively gather specific game criteria into a neat organize fashion. This software would be most useful to consumers who typically purchase several video games a year and/or who currently have a video game collection and wish to organize their collection into a database easily.

Keeping track of video game information using a pencil/paper method is very tedious, which could cause the consumer to make careless errors when recording the information. Using a table style setup on the recording sheet of paper could help reduce human error. In the past there have been database systems that have included barcode scanners to make retrieving information simpler. The flaw with these systems is that in order to retrieve the information an internet connection is needed; another downfall is the price of the software and the barcode scanner which could exceed the budget of the consumer.

The end-users, video game consumers, of this program would enter specified criteria relating to video game information. Input would be given via terminal or command prompt and output would be displayed the same way. User input would include basic game information. Users of these types of systems would be required to input several basic attributes for each video game entered into the database:

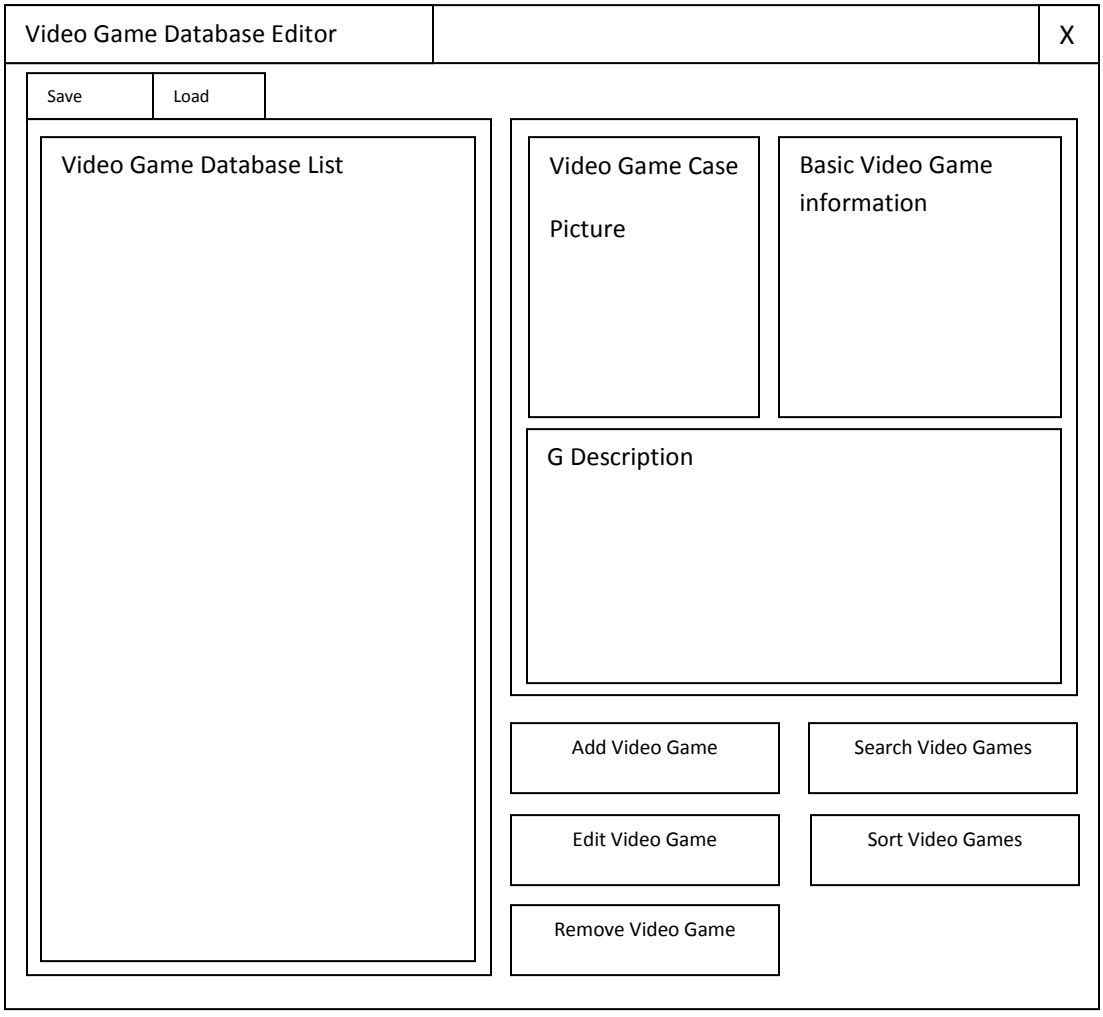
Inputs:

- Title
- Genre
- Console/Platform
- Publisher
- Developer
- Release Date
- ESRB Rating
- Lead Designer(s)
- Average Review Rating

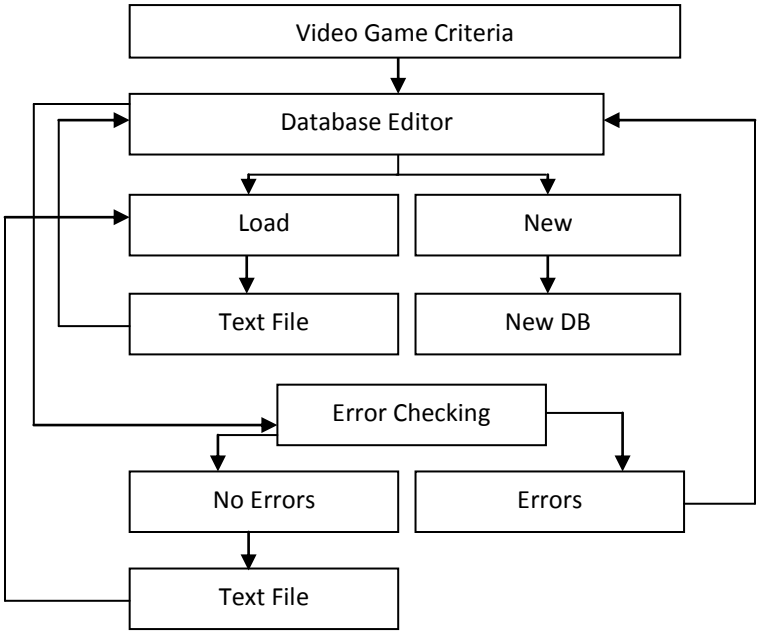
Outputs:

- List of basic video games in database
- List of consoles included in database
- Borrowed or/not borrowed
- Picture of video game case
- Index #
- Description

Some video game databases include user-friendly graphical user interfaces. These interfaces are both visually appealing and helpful to the end-user. The graphical editor helps reduce human and syntax errors. The graphical editor might appear as follows:



This simple user-interface would reduce the frustrations that could be caused by complex interface and incite human error. A simple interface would allow the end-user direct control over the input to the database system. The following flowchart displays possible user action and operation of the system:



A2: Criteria for Success

This section will outline the project goals. Below are the desired behaviors of a successful database software system and the ways in which these behaviors relate to the problem stated previously in the problem analysis section. Also addressed in this section are the usability, limitations, and requirements of the software.

Expected Behaviors:

-Ability to store videogame information (Title, Console, Developer, Release Date, Publisher, ESRB Rating, Producer, Genre, etc.):

- Must be able to sort games alphabetically by the attributes stated above.
- Must allow end-user to edit specific information after the video game has already been added to the database.
- Must be able to quickly search though database for video games that meet specific search criteria
- Must be able to process search requests and return best suitable games to that criteria (i.e. Related Games, Similar Games)

- Ability to read database information from text files:

- To maintain simplicity all database information will be saved and loaded from text files
- Should be able to detect any missing or corrupted information and inform the user.
    - Should be able to classify errors
    - Error messages should be descriptive and concise

-Database Editor:

- Users should be able to add, edit, remove database entries through a graphical or textual user interface
- Through the editor users should have the ability to name and save the database file as well as load databases
- Fast Search
- Ability to sort video game criteria.
- Print a list of all database entries in sequential order

Project Goals:

-Searches queries should return results fast

- Searching through an entire collection of video games can take seconds rather than minutes or longer

-Menu system should be simple and easy to navigate.

- The editor must be able to be used by anyone no matter good their knowledge of computers are

-Error checking to prevent input errors

- Users want the most accurate data included in their database

-Output should be presented in a ordered easy to read fashion

- Users want to be able to easy access and view their database information

Environment Requirement/Restrictions:

-Personal Computer (must meet minimum system requirements for Java Runtime Environment)

- Minimal system memory requirements
- 40KB of free space for database editor
- Additional space is dependent of size of database output files
- Java IDE (i.e. JCreator), or command/prompt

-Updated installation of Java Runtime Environment

Performance:

-For optimal use, a modern computer with an up-to-date OS

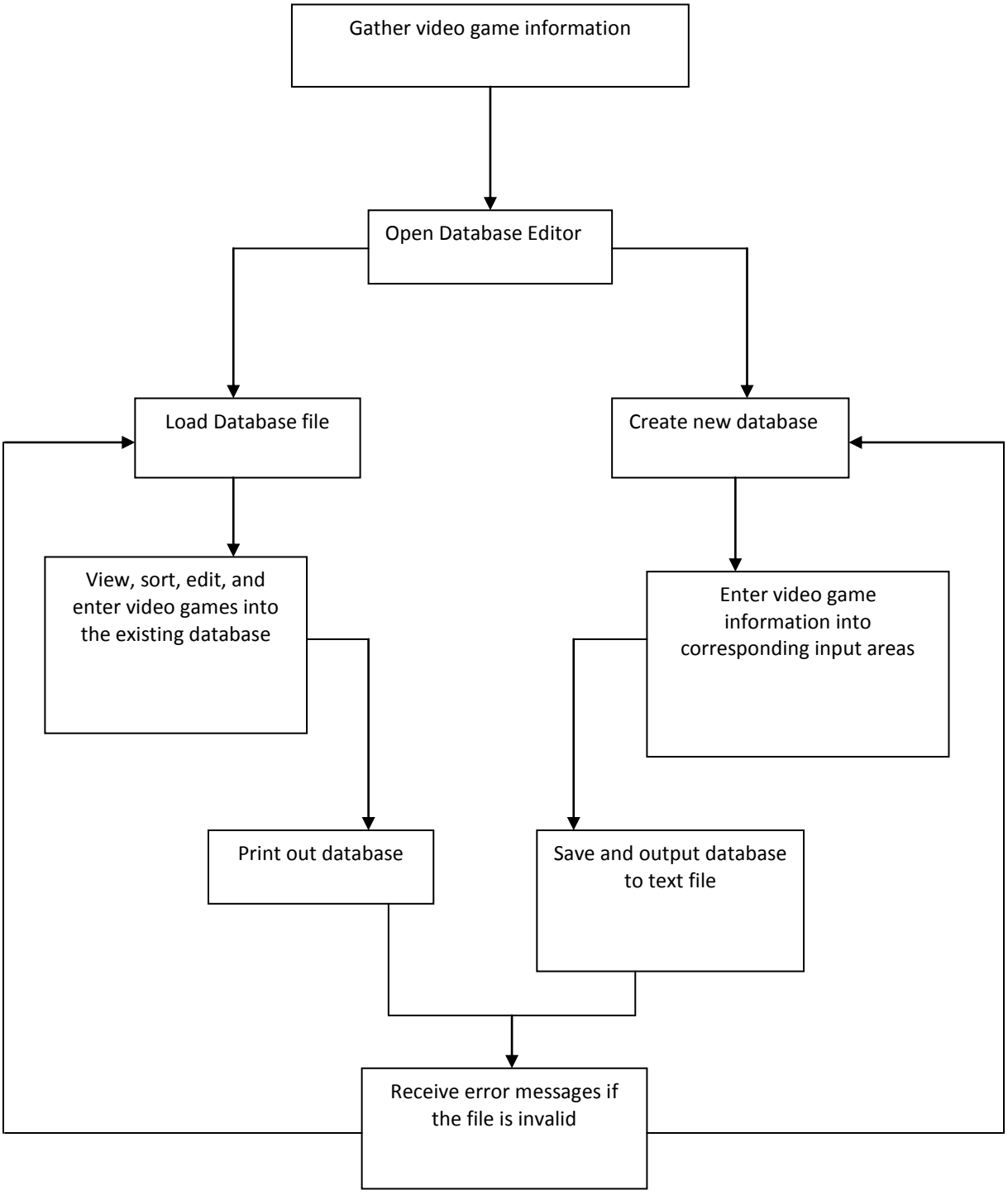
-At least 256MB of memory and 256MB of hard drive space

Relation to Problem

These behaviors should adequately solve the problem stated in section in the problem analysis section. If the program behaves as outlined above, it would greatly assist the user in the organization of a large video collection. The program would allow users to enter video game information into the database which then would be subsequently saved for use at a later time. The program would also allow the user to sort and search the database quickly, allowing the user to locate a game or information on a game much more quickly than traditional methods. The editor will also be able to load up the saved database text documents. With the information available through the edit the user would be able to edit, sort, remove, search, and save databases.

A3: Prototype Solution

The following flowchart details the possible actions of the end-user and the results of each action are given:



Action Descriptions

The different actions from the flowchart are illustrated in the descriptions of the actions below

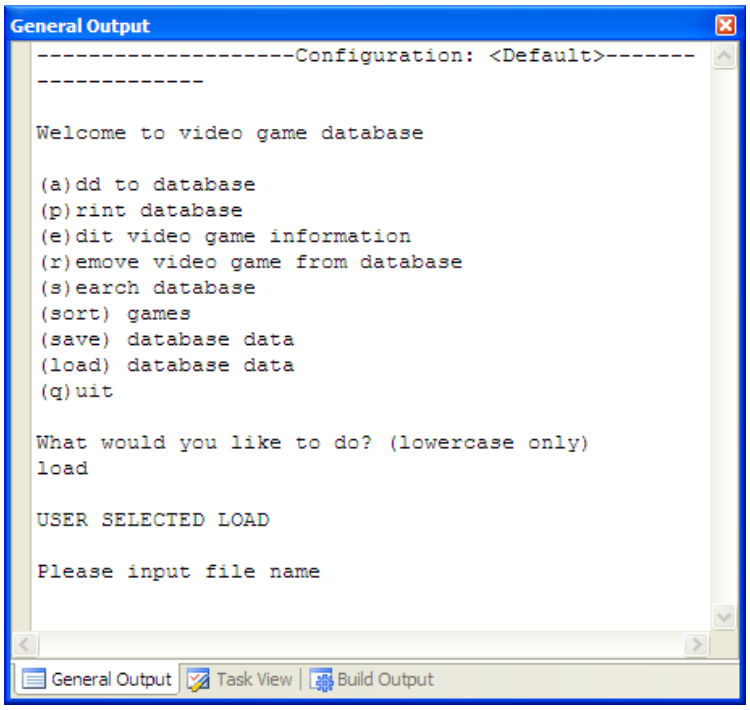
Gather video game information

Using the game cases, manuals, or online resources gather information on the video game (i.e. Title, Platform, Publisher, Developer, Release Date, ESRB Rating, etc.)

Open Database Editor
The database editor would be a java program run from the terminal/console or a java IDE using a textual user interface. The interface would contain the menu options to add, edit, remove, print, and sort the
video game database.

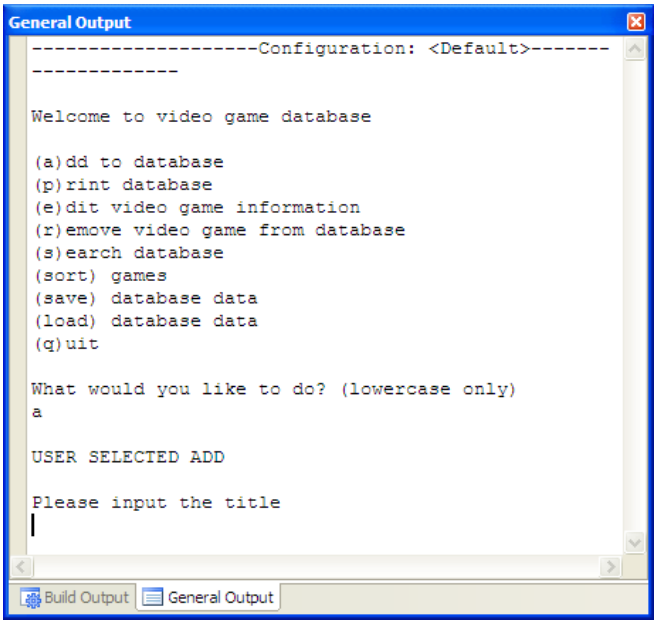
Ex. javac VGDBMain.java
java VGDBMain

Load Database File
If the user already has an existing database on their hard drive, than through the editor they will be able to load the file. In the interface the user will be prompted for the name of the file as well as the location, from there the user would type in the corresponding information to retrieve their database.



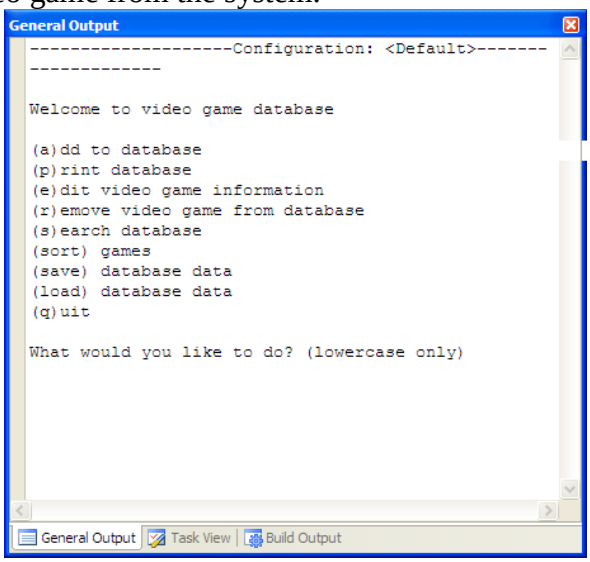
Create New Database

If the user does not already have an existing database then they will be prompted to start inputting video game information into the editor. In the interface the user will be prompted to input a number of basic video game details before the entry is added to the database.



View, sort, edit, and enter video games into the existing database

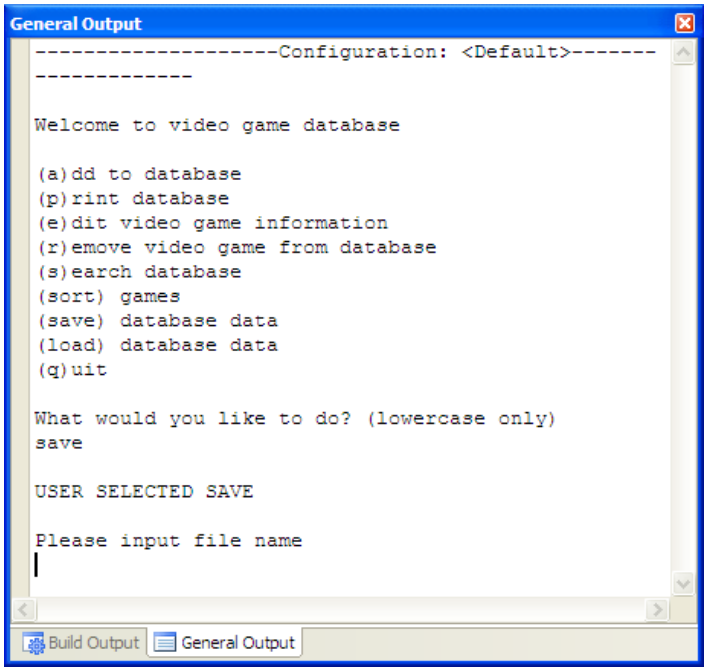
If the database has been loaded or a new one has been created then the user will be prompted with a several options to either add, print, edit, remove, sort, or search the database. Through the sort method the user will be prompted to for a specific attribute to be sorted alphabetically. The edit method will allow the user to edit any specific attribute of existing video games in the database. The remove method will allow the user to remove a specific video game from the system.



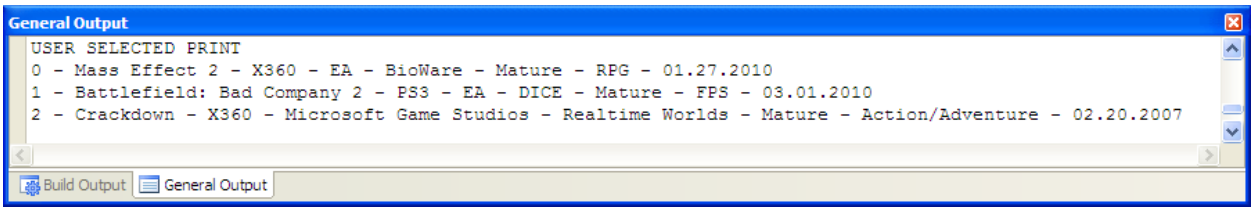
Enter video game information into corresponding input areas
The user will be prompted for several video game details before added the game to the database. The possible attributes are as follows:

- Title
- Platform
- Publisher
- Developer
- Genre
- Release Date
- ESRB Rating

Save and output database to text file
When the save option is selected the user will be prompted for a filename, from there the file will be saved to the local directory and stored there for later use.



Print out database
When the print option is selected the contents of the database are displayed to the user in a sequential order, unless sorted prior to being printed



Receive error messages if the file is invalid
If an invalid file name was entered by the user, or vital inputs are missing from the database entries, an error message will be displayed and the user will be prompted to fix these errors before any saving or other functions are run.
ERROR: Input is Empty
ERROR: Invalid File, File does not exist
ERROR: File is corrupt

User Feedback

Dane Fitzmaurice - I think this is a very well composed structure. I found the descriptions simple, uncluttered, and easy to follow. I also think it would be nice to see internal help menu included for the end-user who have a more difficult time navigating.

Paul Vu - The action-flowchart is comprehensive in the sense that it takes into consideration all of the processes the program will undergo. Each process is thoroughly detailed, with good descriptions of the process itself, and pictures to accompany those descriptions. When the user is prompted to input a filename, they should be told exactly where the file should be located, and whether or not to include the file extension (.txt, etc). Overall, the program is well organized, and end-users will be able to use it without any complications.

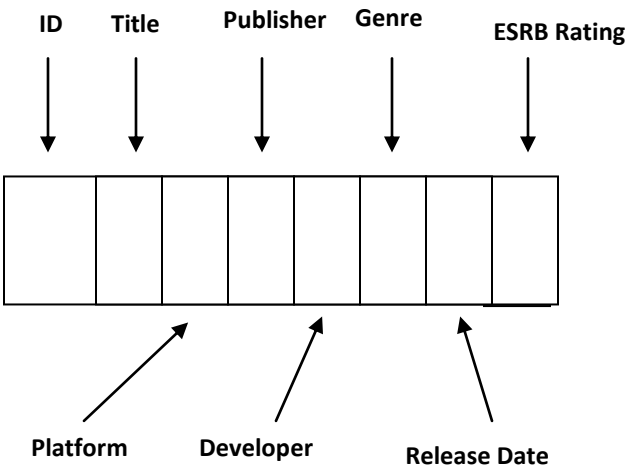
B1: Data Structures

The elementary class to be used in this particular solution is the VideoGame class. It will contain the data identifying and describing a specific videogame in the database. Video games in the database will be linked through references in a linked list structure called a DB. The DB is accessed through head/tail attributes. DB will act as a stack where VideoGame class elements will be added to either the head/tail or in between. The DB will also include sorting, adding, and removing methods, these method descriptions are located in the DB section below. After being processed through the DB the videogames will then be accessed through VGDB. The VGDB compiles and simplifies all the methods located within the DB in order to be used in VGDBMain.

VideoGame

The VideoGame class consists of several private attributes, a default constructor, and several setters and getters.

In the memory, this is how the VideoGame class would appear:



ID: Specific identification number given to the game by order it was added in.

Title: Title of video game

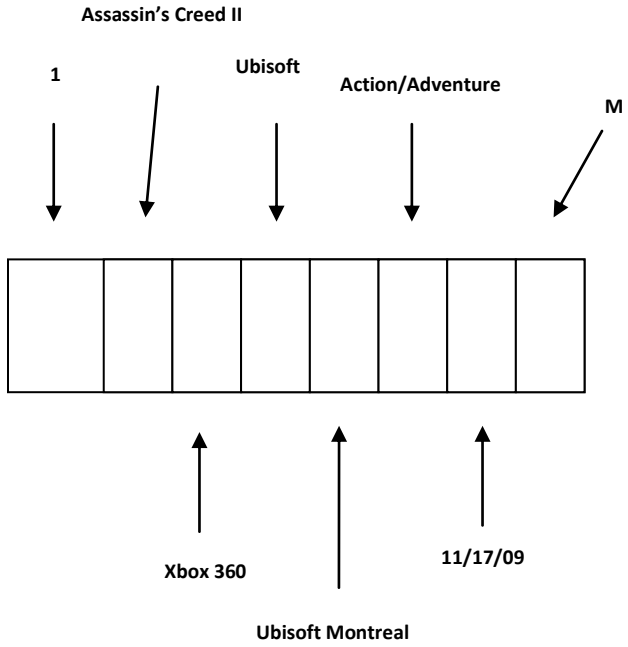
Platform: Console for which video game is for.

Publisher: Publisher of video game

ID is stored as and int, giving the ability to sort by last game or first game added to database.

Title, Platform, Publisher, Developer, Genre, Release Date, and ESRB Rating are all determined by end-user input. These attributes are all sortable elements and are used for identification and informational purposes.

Example VideoGame Class information:



ID: 1

Title: Assassin's Creed II

Platform: Xbox 360

Publisher: Ubisoft

Developer: Ubisoft Montreal

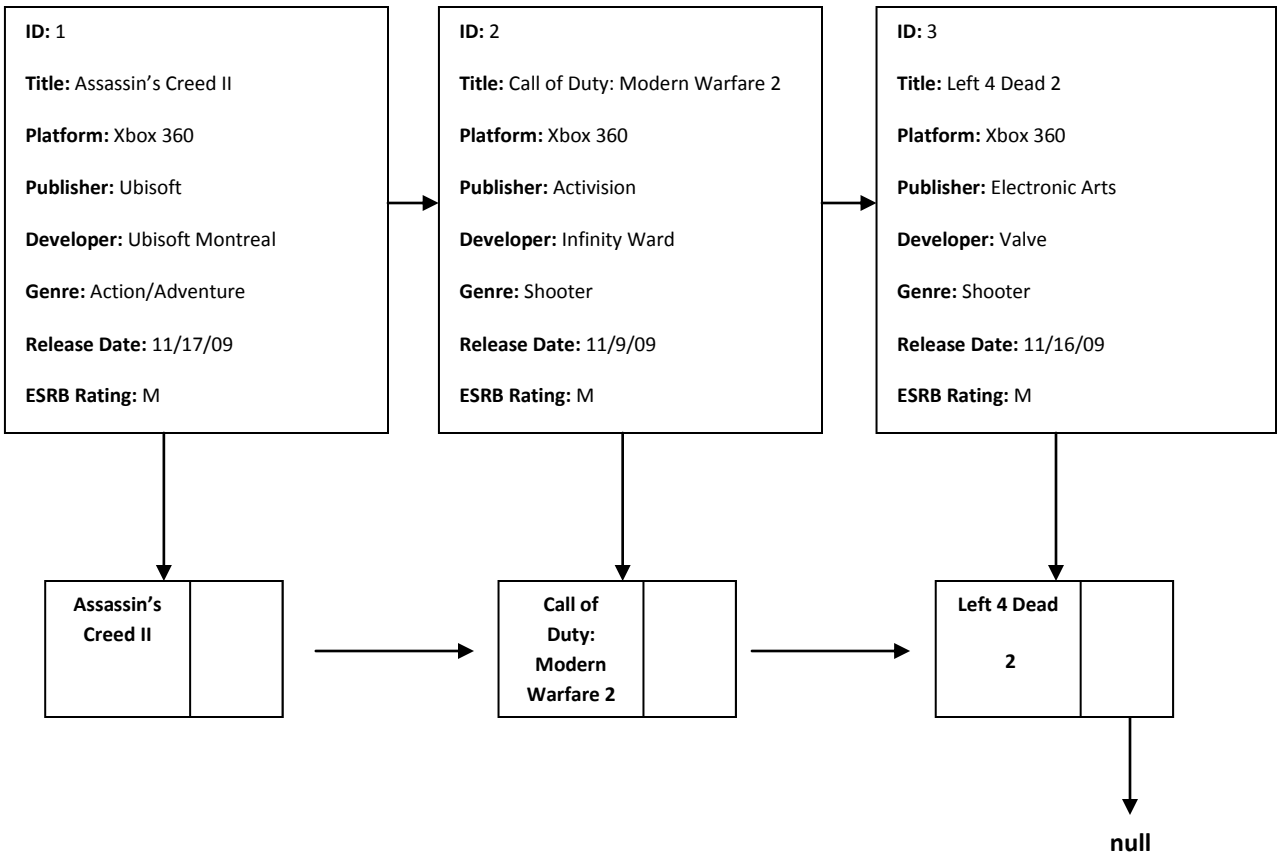
Genre: Action/Adventure

Release Date: 11/17/09

ESRB Rating: M

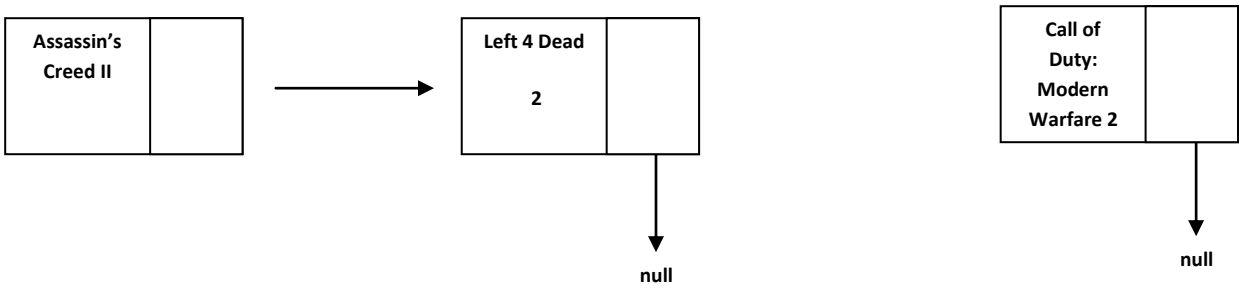
DB

The DB class represents a linked list including a head, tail, and nodes.

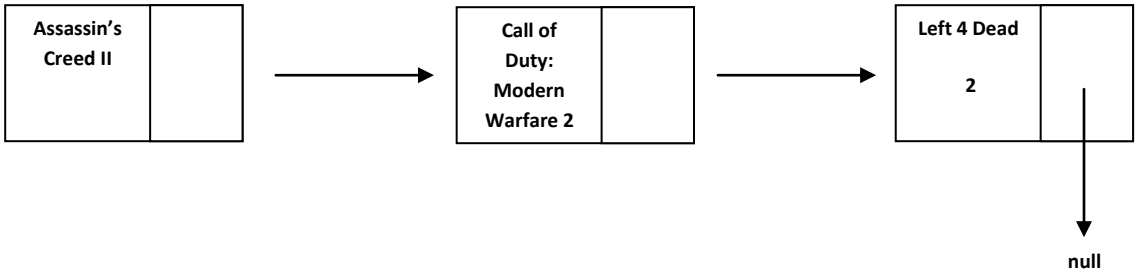


Adding videogames to a DB is a simple procedure only requiring and insertion point and a new VideoGame instance:

Before Inserting:

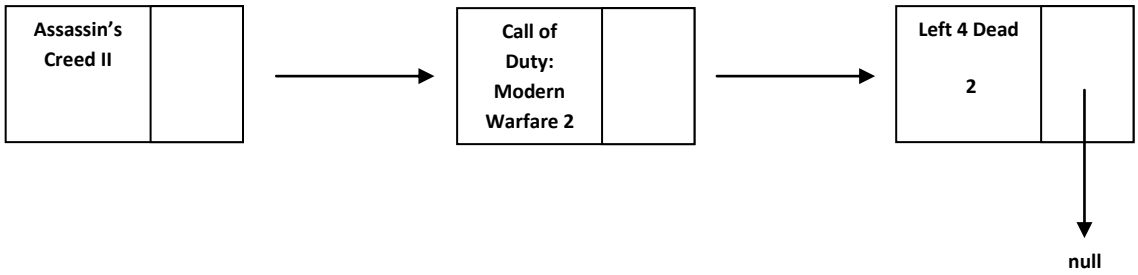


After Inserting:

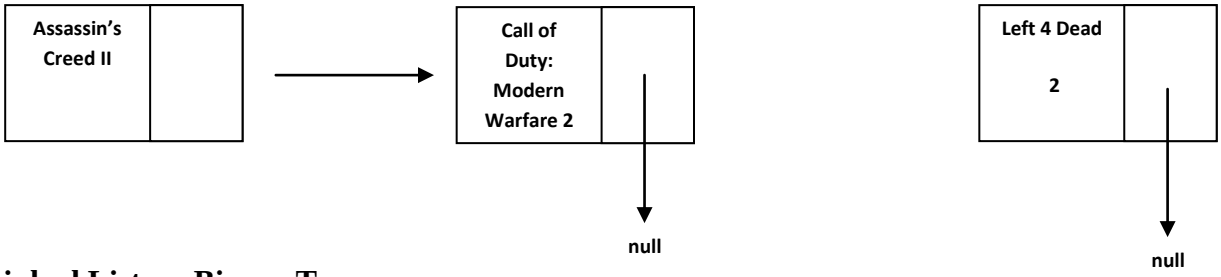


Removing videogames from a DB is just as simple as adding the only requirement is a point to remove from.

Before Removing:



After Removing:



Linked List vs. Binary Tree

A binary tree could be used as a solution to the problem, being that it has a faster searching structure and less comparison need to be done compared to that of a linked list. But there is no need for inherent ordering. With this solution the ordering would eventually need to be overwritten to provide the end-user with all options and capabilities of the database.

B2: Algorithms

Adding/Removing Games from Database

These algorithms will be used to add and remove games from the game database

addGames

Name:	addGames
Description:	Adds game to database
Parameters:	Strings vgTitle, vgPlatform, vgPublisher, vgDeveloper, vgGenre, vgRelease, vgESRB
Return Value:	VGDB linked list with game node added
Pre-Condition:	Video game is not in VGDB linked list
Post-Condition:	Game node in VGDB linked list
Pseudocode:	1. Check to see if VGDB linked list is empty 2. If VGDB linked list is not empty, add game 3. Return VGDB linked list with game added

removeGame

Name:	removeGame
Description:	Removes game from database
Parameters:	A integer count
Return Value:	VGDB linked list with game node removed
Pre-Condition:	Video game is in VGDB linked list
Post-Condition:	Game node not in VGDB linked list
Pseudocode:	1. Check to see if count matches any of the game's ID number in VGDB linked list 2. If count does equal the ID of the game, remove game 3. Return VGDB linked list with game removed

Printing from Database
These algorithms will be used to print out a list of entries from the database

getGame

Name:	getGame
Description:	Retrieves game from the database
Parameters:	A integer index
Return Value:	Returns the game node that is located at that index
Pre-Condition:	Game node with index is in database
Post-Condition:	Game node is returned
Pseudocode:	1. Check to see if index equals count 2. If index does equal count, return game node 3. Return game node

print

Name:	print
Description:	Prints out the database in sequential order to be view by user
Parameters:	None
Return Value:	List database entries in sequential or sorted order
Pre-Condition:	Database has not yet been printed
Post-Condition:	Database has been printed
Pseudocode:	1. Temp equals head 2. While temp is not equal to null, print temp.myGame 3. Temp equals temp.next

Saving Database
These algorithms will be used to save the database

dbWrite

Name:	dbWrite
Description:	Saves database to a text file
Parameters:	A string fileName
Return Value:	Database saved as a text file
Pre-Condition:	Database has not yet been saved
Post-Condition:	Database has been saved
Pseudocode:	1. Write linked list size to text file 2. Write all attributes of all linked list nodes to file 3. Close file

Searching Database
These algorithms will be used to search through the database for a specific entry

find

Name:	find
Description:	Searches through database locating game node that matches specific criteria
Parameters:	A string Title, A string platform
Return Value:	Game node that matches criteria
Pre-Condition:	Game node has not been found
Post-Condition:	Game node has been found
Pseudocode:	1. Prompt user for title and platform 2. Traverse linked list looking for game node that matches both title and platform 3. If match is found, return game node 4. Return game node

Sorting Database
These algorithms will be used to sort the database

alphaSort

Name:	alphaSort
Description:	Sort database alphabetically by attribute specified by user
Parameters:	A string sortInput
Return Value:	A database sorted by attribute specified by user
Pre-Condition:	Database is not sorted by attribute specified by user
Post-Condition:	Database is sorted by attribute specified by user
Pseudocode:	1. Check sortInput, sort by corresponding if statement 2. Compare adjacent node attribute values, return node which has attribute closer to beginning of alphabet 3. Return sorted database

Removing from database
These algorithms will be used to remove from the database

idSearch

Name:	idSearch
Description:	Searches through database to see determine what index the game node with the corresponding id number is located
Parameters:	A int ID
Return Value:	Return the index of the game node
Pre-Condition:	Index of game node with corresponding ID has not been returned
Post-Condition:	Index of game node with corresponding ID has been returned
Pseudocode:	1. Traverse linked list 2. Compare ID attribute to int ID 3. If ID attribute equals int ID, return game node 4. Return game node

reNum

Name:	reNum
Description:	Renumbers all game nodes, usually initialized after a game node has been removed
Parameters:	none
Return Value:	Renumbered database
Pre-Condition:	Database has not been renumbered
Post-Condition:	Database has been renumbered
Pseudocode:	1. Int ID equals 0 2. Traverse linked list, setID equal to int ID 3. Add 1 to int ID 4. Return renumbered database

Error Checking
These algorithms will be used to check for input errors in the database

getString

Name:	getString
Description:	Error checking method, checks to see if input is empty
Parameters:	A string userPrompt
Return Value:	Return userInput, return error message
Pre-Condition:	userInput has not been check to see if it is empty
Post-Condition:	userInput has been check to see if it is empty
Pseudocode:	1. Print userPrompt 2. If userInput is empty, return error message 3. If userInput is not empty, return userInput

getNPString

Name:	getNPString
Description:	Error checking method, checks to see if input is empty. Does not require userPrompt
Parameters:	None
Return Value:	Return userInput, return error message
Pre-Condition:	userInput has not been check to see if it is empty
Post-Condition:	userInput has been check to see if it is empty
Pseudocode:	1. If userInput is empty, return error message 2. If userInput is not empty, return userInput

getRD

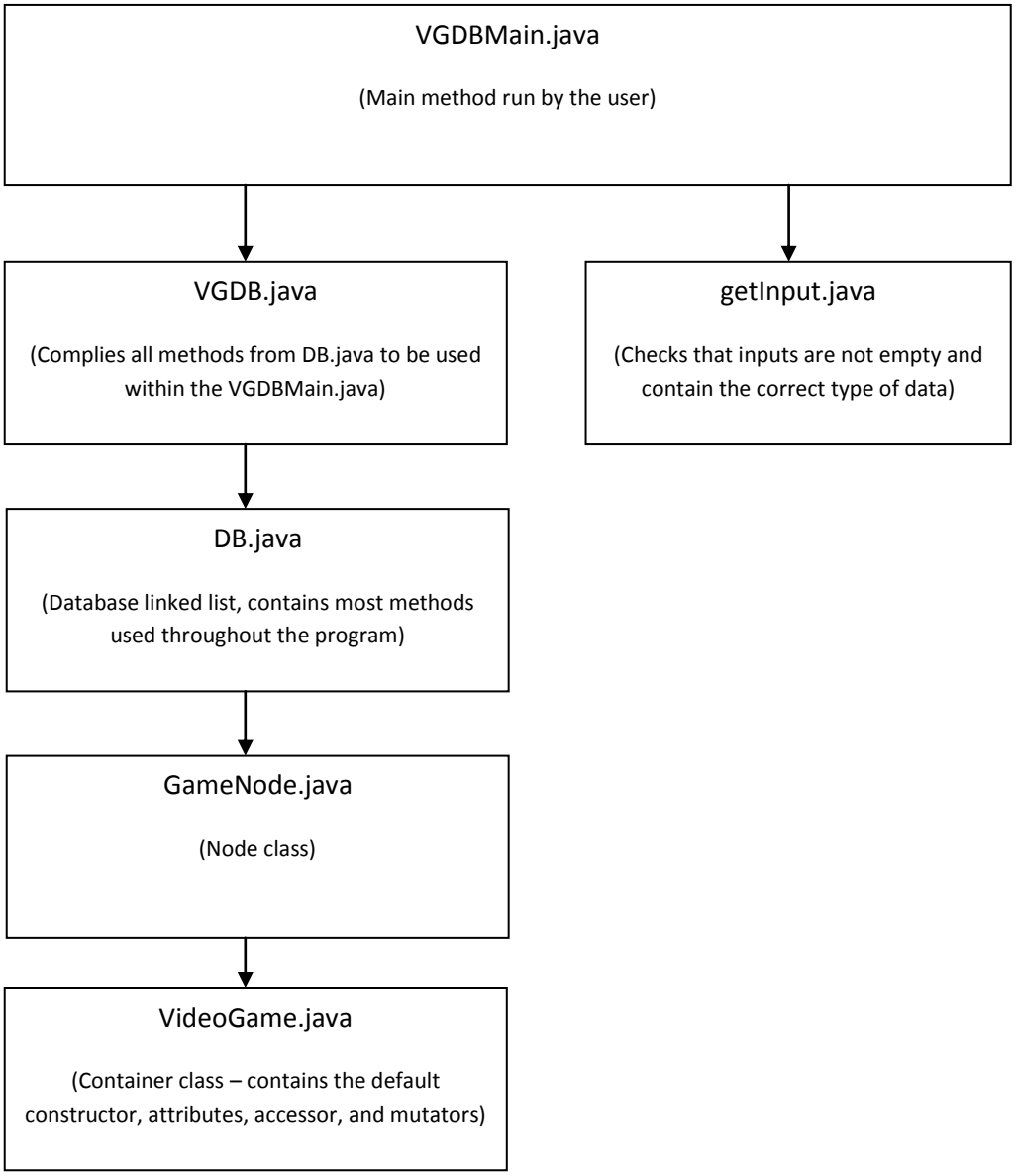
Name:	getRD
Description:	Error checking method, checks to see if release date is in accurate date, format, and contains only numbers.
Parameters:	A string userPrompt
Return Value:	Return userInput, return error message
Pre-Condition:	userInput has not been check to see if it is empty, formatted correctly, and contains only numbers
Post-Condition:	userInput has been check to see if it is empty, formatted correctly, and contains only numbers
Pseudocode:	1. Print userPrompt 4. If userInput is empty, return error message 2. Else if userInput is all numbers 3. Check to see if month and day are within limits of calendar months and days 4. Return userInput

getNPRD

Name:	getNPRD
Description:	Error checking method, checks to see if release date is in accurate date, format, and contains only numbers. Does not require userPrompt
Parameters:	None
Return Value:	Return userInput, return error message
Pre-Condition:	userInput has not been check to see if it is empty, formatted correctly, and contains only numbers
Post-Condition:	userInput has been check to see if it is empty, formatted correctly, and contains only numbers
Pseudocode:	1. Print userPrompt 2. If userInput is empty, return error message 3. Else if userInput is all numbers 4. Check to see if month and day are within limits of calendar months and days 5. Return userInput

B3: Modular Organization

The following hierarchy diagram shows the solution and its modules



VGDBMain.java
This class is a main menu in which the end-user can edit the database by adding, removing, sorting, or searching. It contains no methods. It provides a user-friendly text-based interface that allows the user to perform certain functions to the video game database.

VGDB.java
VGDB.java compiles and simplifies all the methods located within the DB in order to be used in VGDBMain.

Attributes

- myGames – a DB

Methods

- Size – returns size
- addGames – adds GameNode to linked list
- removeGame – removes GameNode from linked list
- getGame – returns GameNode
- print – returns linked list
- dbWrite – saves linked list to text file
- find – Traverses linked list looking for a GameNode that matches user specific criteria
- alphaSort – Sorts the linked list alphabetically or sequentially based on user defined criteria
- idSearch – Traverses the linked list to find the index of GameNode with user specified ID
- reNum – Renumbers all IDs in linked list to a sequential order

DB.java

This is the linked list class, this file houses all the primary functions of the database creator/editor.

Attributes

- Head – head of linked list
- Tail – tail of linked list
- Size – size of linked list

Methods

- Size – returns size
- isEmpty –checks to see if linked list is empty
- add – adds GameNode to linked list
- remove – removes GameNode from linked list
- getGame – returns GameNode
- print – returns linked list
- dbWrite – saves linked list to text file
- find – Traverses linked list looking for a GameNode that matches user specific criteria
- alphaSort – Sorts the linked list alphabetically or sequentially based on user defined criteria
- idSearch – Traverses the linked list to find the index of GameNode with user specified ID
- reNum – Renumbers all IDs in linked list to a sequential order

GameNode.java

This is the node class that contains each video game in the database link list. There are no methods located in this class.

Attributes

- myGame –set to null
- GameNode next – set to null

VideoGame.java

This is the container class that sets all the attributes that each GameNode consists of. The are two special methods located in this file: mutators (setters) and accessors (getters). These allow the retrieval of the attributes from other local files, as well as the ability to overwrite or send new attributes.

Attributes

- Title
- Platform
- Publisher
- Developer
- Genre
- Release Date
- ESRB

Methods

- setTitle – Sets Title
- setPlatform – Sets Platform
- setPublisher – Sets Publisher
- setDeveloper – Sets Developer
- setGenre – Sets Genre
- setRelease – Sets Release Date
- setESRB – Sets ESRB Rating
- getTitle – Gets Title
- getPlatform – Gets Platform
- getPublisher – Gets Publisher
- getDeveloper – Gets Developer
- getGenre – Gets Genre
- getRelease – Gets Release Date
- getESRB – Gets ESRB Rating

getInput.java

This is the error checking class. In this file there are four methods all of which check to make sure there are no possible errors when running through the editor and inputting in information

Methods

- getString – prints prompt, returns error message if input is empty
- getNPString – does not print a prompt, returns error message if input is empty
- getRD – prints prompt, returns error message if input is empty, format is incorrect, not all characters are numbers, or if month and day do not match any calendar date
- getNPRD – does not print prompt, returns error message if input is empty, format is incorrect, not all characters are numbers, or if month and day do not match any calendar date

C1: Good Programming Style

```
1   /* Project Name: IB Computer Science Dossier
2    * Program Name: Video Game Database Creator/Editor
3    * File Name: VideoGame.java
4    * Author: Sean Michael Hoffman
5    * Date: March 23, 2010
6    * School: Washington-Lee High School
7    * IDE: JCreator
8    */
9
10
11  public class VideoGame
12  {
13          // attributes
14          private int vgID = 0;
15          private String vgTitle = "";
16          private String vgPlatform = "";
17          private String vgPublisher = "";
18          private String vgDeveloper = "";
19          private String vgGenre = "";
20          private String vgRelease = "";
21          private String vgESRB = "";
22
23
24          // default constructor - initializes default attr values
25          public VideoGame()
26          {
27          }
28
29          // specific constructor
30          public VideoGame(int newID, String newTitle, String newPlatform, String newPublisher, String
31          newDeveloper, String newGenre, String newRelease, String newESRB )
32          {
33                  vgID = newID;
34                  vgTitle = newTitle;
35                  vgPlatform = newPlatform;
36                  vgPublisher = newPublisher;
37                  vgDeveloper = newDeveloper;
38                  vgGenre = newGenre;
39                  vgRelease = newRelease;
40                  vgESRB = newESRB;
41          }
42
43          // getters (accessors)
44          public int getID()
45          {
46                  return vgID;
47          }
48          public String getTitle()
49          {
50                  return vgTitle;
51          }
52          public String getPlatform()
53          {
```

```
54                  return vgPlatform;
55          }
56          public String getPublisher()
57          {
58                  return vgPublisher;
59          }
60          public String getDeveloper()
61          {
62                  return vgDeveloper;
63          }
64          public String getGenre()
65          {
66                  return vgGenre;
67          }
68          public String getRelease()
69          {
70                  return vgRelease;
71          }
72          public String getESRB()
73          {
74                  return vgESRB;
75          }
76
77          // setters (mutators)
78          public void setID(int newID)
79          {
80                  vgID = newID;
81          }
82          public void setTitle(String newTitle)
83          {
84                  vgTitle = newTitle;
85          }
86          public void setPlatform(String newPlatform)
87          {
88                  vgPlatform = newPlatform;
89          }
90          public void setPublisher(String newPublisher)
91          {
92                  vgPublisher = newPublisher;
93          }
94          public void setDeveloper(String newDeveloper)
95          {
96                  vgDeveloper = newDeveloper;
97          }
98          public void setGenre(String newGenre)
99          {
100                 vgGenre = newGenre;
101         }
102         public void setRelease (String newRelease)
103         {
104                 vgRelease = newRelease;
105         }
106         public void setESRB (String newESRB)
107         {
108                 vgESRB = newESRB;
109         }
```

```
110		public String toString()
111		{
112			return vgID + " - " + vgTitle + " - " + vgPlatform + " - " + vgPublisher + " - " + vgDeveloper + " -
113	" + vgESRB + " - " + vgGenre + " - " + vgRelease;
114		}
115	}
```

```
1   /* Project Name: IB Computer Science Dossier
2    * Program Name: Video Game Database Creator/Editor
3    * File Name: GameNode.java
4    * Author: Sean Michael Hoffman
5    * Date: March 23, 2010
6    * School: Washington-Lee High School
7    * IDE: JCreator
8    */
9
10  public class GameNode
11  {
12          public VideoGame myGame = null;
13          public GameNode next = null;
14
15          //default constructor
16          public GameNode ()
17          {
18          }
19  }
```

```java
/* Project Name: IB Computer Science Dossier
 * Program Name: Video Game Database Creator/Editor
 * File Name: getInput.java
 * Author: Sean Michael Hoffman
 * Date: March 23, 2010
 * School: Washington-Lee High School
 * IDE: JCreator
 */
import java.util.Scanner;
public class getInput
{
	//getString(String userPrompt) prints out the string userPrompt and makes sure the user input is not empty
	public static String getString(String userPrompt)
	{
		Scanner scan = new Scanner(System.in);//initialize scanner
		while (true)
		{
			System.out.println(userPrompt);//print prompt
			String userInput = scan.nextLine();//userInput equals input

			if (userInput.length() == 0)//input is empty, return error
			{
				System.out.println("\nERROR: Invalid Input");
				System.out.println("Input was empty");
				continue;
			}
			else
			{
				return userInput;//return input
			}
		}

	}
	//getNPString() Makes sure the user input is not empty
	public static String getNPString()
	{
		Scanner scan = new Scanner(System.in);//initialize scanner
		while (true)
		{
			String userInput = scan.nextLine();//userInput equals input

			if (userInput.length() == 0)//input is empty, return error
			{
				System.out.println("\nERROR: Invalid Input");
				System.out.println("Input was empty");
				continue;
			}
			else
			{
				return userInput;
			}
		}

	}
	//getRD(sting userPrompt) prints out the string userPrompt, and makes sure the input is not empty, and
	there are no format, range, or type errors in the release date attribute
```

```
57          public static String getRD(String userPrompt)
58          {
59                  Scanner scan = new Scanner(System.in);//initialize scanner
60                  while(true)
61                  {
62                          System.out.println(userPrompt);
63                          String userInput = scan.nextLine();//userInput equals input
64
65                          if (userInput.length() == 0)//input is empty, return error
66                          {
67                                  System.out.println("\nERROR: Invalid Input");
68                                  continue;
69                          }
70                          else
71                          {
72                                  String numCheck = userInput.replace(".", "");//remove periods from string
73                                  boolean invInput = false;
74                                  for (int i = 0; i < numCheck.length(); i++)
75                                          if (!(Character.isDigit(numCheck.charAt(i))))//check to see if all char
76                                          as digits
77                                          {
78                                                  invInput = true;
79                                                  break;
80                                          }
81                                  if(invInput)//if invInput is true, return error
82                                  {
83                                          System.out.println("\nERROR: Invalid Input");
84                                          continue;
85                                  }
86                                  Scanner scan2 = new Scanner(userInput).useDelimiter("\\.");
87                                  //initialize scanner, set seperator to period
88
89                                  int month = scan2.nextInt();
90                                  int day = scan2.nextInt();
91
92                                  if (month == 0 || month > 12)//checks if month falls in between 1-12, return
93                                  error if not
94                                  {
95                                          System.out.println("\nERROR: Invalid Input");
96                                          System.out.println("You have exceeded the number of months in a
97                                          year. Try Again.");
98                                          continue;
99                                  }
100                                 else if (day == 0 || day > 31)//check if day falls in between 1-31, return error if
101                                 not
102                                 {
103                                         System.out.println("\nERROR: Invalid Input");
104                                         System.out.println("You have exceeded the number of days in a month.
105                                         Try Again.");
106                                         continue;
107                                 }
108                                 else
109                                         return userInput;
110                         }
111                 }
112         }
```

```
113         //getNPRD() Makes sure the input is not empty, checks for format, range, or type errors
114         public static String getNPRD()
115         {
116                 Scanner scan = new Scanner(System.in);//initialize scanner
117                 while(true)
118                 {
119                         String userInput = scan.nextLine();//userInput equals input
120
121                         if (userInput.length() == 0)//input is empty, return error
122                         {
123                                 System.out.println("\nERROR: Invalid Input");
124                                 continue;
125                         }
126                         else
127                         {
128                                 String numCheck = userInput.replace(".", "");//remove periods from string
129                                 boolean invInput = false;
130                                 for (int i = 0; i < numCheck.length(); i++)
131                                         if (!(Character.isDigit(numCheck.charAt(i))))//check to see if all char
132                                         as digits
133                                         {
134                                                 invInput = true;
135                                                 break;
136                                         }
137                                 if(invInput)//if invInput is true, return error
138                                 {
139                                         System.out.println("\nERROR: Invalid Input");
140                                         continue;
141                                 }
142                                 Scanner scan2 = new Scanner(userInput).useDelimiter("\\.");
143
144                                 int month = scan2.nextInt();
145                                 int day = scan2.nextInt();
146
147                                 if (month == 0 || month > 12)//checks if month falls in between 1-12, return
148                                 error if not
149                                 {
150                                         System.out.println("\nERROR: Invalid Input");
151                                         System.out.println("You have exceeded the number of months in a
152                                         year. Try Again.");
153                                         continue;
154                                 }
155                                 else if (day == 0 || day > 31)//check if day falls in between 1-31, return error if
156                                 not
157                                 {
158                                         System.out.println("\nERROR: Invalid Input");
159                                         System.out.println("You have exceeded the number of days in a month.
160                                         Try Again.");
161                                         continue;
162                                 }
163                                 else
164                                         return userInput;
165                         }
166                 }
167         }
168 }
```

```
1   /* Project Name: IB Computer Science Dossier
2    * Program Name: Video Game Database Creator/Editor
3    * File Name: DB.java
4    * Author: Sean Michael Hoffman
5    * Date: March 23, 2010
6    * School: Washington-Lee High School
7    * IDE: JCreator
8    */
9   import java.io.*;
10  public class DB
11  {
12          private static BufferedReader stdin =
13      new BufferedReader( new InputStreamReader( System.in ) );
14          // attributes
15          private GameNode head = null;
16          private GameNode tail = null;
17          private int size = 0;
18          // default constructor
19          public DB()
20          {
21          }
22          public int size()
23          {
24                  return size;
25          }
26
27          // isEmpty() returns true if the LinkedList is empty, and false otherwise
28          public boolean isEmpty()
29          {
30                  return size == 0;
31                  //return head == null;
32          }
33
34          // add() adds a new number to the end of the LinkedList
35          public void add(VideoGame myGame)
36          {
37
38                  //1st case - adding to empty LL
39                  if (isEmpty())
40                  {
41                          GameNode vgNode = new GameNode();
42                          vgNode.myGame = myGame;
43                          head = vgNode;
44                          tail = vgNode;
45                          size++;
46                  }
47                  //2nd case - adding to non-empty LL
48                  else if (!isEmpty())
49                  {
50                          GameNode vgNode = new GameNode();
51                          vgNode.myGame = myGame;
52                          tail.next = vgNode;
53                          tail = vgNode;
54                          size++;
55                  }
56          }
```

```
57
58          // remove(int index) - removes a Node and number at a particular index
59          public VideoGame remove(int index)
60          {
61                  //0) removing from index out of bounds
62                  if (index >= size || index < 0)
63                  {
64                          return null;
65                  }
66
67                  //1) remove from empty LL - return error code
68                  if (isEmpty())
69                  {
70                          return null;
71                  }
72
73                  //2) remove last Node from LL
74                  if (index == 0 && head == tail)      // if (size == 1)
75                  {
76                          VideoGame vgNode = head.myGame;
77                          head = null;
78                          tail = null;
79                          size--;
80                          return vgNode;
81                  }
82
83                  //3) remove from the middle (b/w head and tail) of the LL
84                  if (index != 0 && index != size -1)
85                  {
86                          int count = 0;
87                          GameNode previous = head;
88                          for (GameNode current = head; current != null; current = current.next)
89                          {
90                                  //if you are at the correct current, set the previous next to the current's next
91                                  if (count == index)
92                                  {
93                                          VideoGame x = current.myGame;
94                                          previous.next = current.next;
95                                          size--;
96                                          return x;
97                                  }
98
99                                  previous = current;
100                                 count++;
101                         }
102                 }
103
104                 //4) remove from the head
105                 if (index == 0 && head != tail)
106                 {
107                         VideoGame vgNode = head.myGame;
108                         head = head.next;
109                         size--;
110                         return vgNode;
111                 }
112
```

```
113                 //5) remove from the tail
114                 if ( index == size - 1 && size > 1)
115                 {
116                         VideoGame vgNode = tail.myGame;
117
118                         for (GameNode current = head; current != null; current = current.next)
119                         {
120                                 if (current.next == tail)
121                                 {
122                                         tail = current;
123                                         tail.next = null;    //current.next = null;
124                                         size--;
125                                         return vgNode;
126                                 }
127                         }
128                         return vgNode;
129                 }
130                 return null;
131         }
132
133                 //add myGame to the location index
134         public void add(VideoGame myGame, int index)
135         {
136                 GameNode vgNode = new GameNode();
137                 vgNode.myGame = myGame;
138                 //0) adding to empty LL
139                 if(isEmpty())
140                 {
141                         head = vgNode;
142                         tail = vgNode;
143                         size++;
144                 }
145                 //1) adding to the head (beginning)
146                 if (index == 0)
147                 {
148                         vgNode.next = head;
149                         head = vgNode;
150                         size++;
151                 }
152                 //2) adding to the tail (end)
153                 if (index == size-1)
154                 {
155                         tail.next = vgNode;
156                         tail = vgNode;
157                         size++;
158                 }
159                 //3) adding to the middle
160                 if (index != 0 || index != size-1)
161                 {
162                         int count = 0;
163                         GameNode previous = head;
164                         for (GameNode current = head.next; current != null; current = current.next)
165                         {
166
167                                 if (count == index)
168                                 {
```

```
169                                         previous.next = vgNode;
170                                         vgNode.next = current;
171                                         size++;
172                                         break;
173                                 }
174                                 count++;
175                                 previous = previous.next;
176                         }
177                 }
178                 //4) adding to an index that is out of bounds
179                 if(index >= size || index < 0)
180                 {
181                         System.out.println ("That index is out of bounds!");
182                 }
183         }
184
185         //getGame(int index) returns the number found in the Node at the specified index
186         public VideoGame getGame(int index)
187         {
188                 int count = 0;
189                 for (GameNode temp = head; temp!= null; temp = temp.next)
190                 {
191                         if (count == index)
192                         {
193                                 return temp.myGame;
194                         }
195                         count ++;
196                 }
197                 return null;
198         }
199         //print() traverses the linked list and prints out the attributes in each Node
200         public void print ()
201         {
202                 for (GameNode temp = head; temp!= null; temp = temp.next)
203                 {
204                         System.out.println (temp.myGame);
205                 }
206         }
207         //dbWrite(String fileName) traverses the linked list writes all attributes to the file specified and created by
208         the user
209         public void dbWrite(String fileName)
210         {
211                 try
212                 {
213                         //initialize randomaccessfile, readable and writable
214                         RandomAccessFile file = new RandomAccessFile(fileName + ".txt", "rw");
215                         String fileSize = Integer.toString(size);
216                         file.writeUTF(fileSize);//write linked list size to top of file
217                         for (GameNode temp = head; temp!= null; temp = temp.next)//traverse through linked list
218                         {       //write all attributes to file
219                                 file.writeUTF(Integer.toString(temp.myGame.getID()));
220                                 file.writeUTF(temp.myGame.getTitle());
221                                 file.writeUTF(temp.myGame.getPlatform());
222                                 file.writeUTF(temp.myGame.getPublisher());
223                                 file.writeUTF(temp.myGame.getDeveloper());
224                                 file.writeUTF(temp.myGame.getGenre());
```

```
225                                     file.writeUTF(temp.myGame.getRelease());
226                                     file.writeUTF(temp.myGame.getESRB());
227                             }
228                             file.close();
229                     }
230                     catch(IOException e) //return error if problem is detected
231             {
232                     System.out.println(e.toString());
233             }
234             }
235             //find(String title, String platform) traverses the linked list looking for any nodes that match the criteria
236             specified by the user
237             public int find(String title, String platform)
238             {
239                     int count = 0;
240                     for (GameNode temp = head; temp!= null; temp = temp.next)//traverse through linked list
241                     {
242                             if (temp.myGame.getTitle().equals(title) &&
243                             temp.myGame.getPlatform().equals(platform))
244                             {
245                                     return count;
246                             }
247                             count ++;
248                     }
249                     return -1;
250             }
251             //alphaSort(String sortInput) traverses the linked list sorting alphabetically by the attribute specified by the
252             user
253             public void alphaSort(String sortInput)
254             {
255                     if (sortInput.equals("id"))
256                     {
257                             GameNode minNode;
258                             for (GameNode front = head; front != null; front = front.next)//traverse through linked list
259                             {
260                                     minNode = front;
261                                     for(GameNode current = front; current != null; current = current.next)//traverse
262                                     through linked list
263                                     {
264                                             If
265                                             (Integer.toString(minNode.myGame.getID()).compareTo(Integer.toStri
266                                             ng(current.myGame.getID())) > 0)
267                                             {//compare attributes
268                                                     minNode = current;
269                                             }
270                                     }
271                                     //swap
272                                     VideoGame temp = front.myGame;
273                                     front.myGame = minNode.myGame;
274                                     minNode.myGame = temp;
275                             }
276                      }
277                     if (sortInput.equals("t"))
278                     {
279                             GameNode minNode;
280                             for (GameNode front = head; front != null; front = front.next)//traverse through linked list
```

```
281                     {
282                             minNode = front;
283                             for(GameNode current = front; current != null; current = current.next)//traverse
284                             through linked list
285                             {
286                                     if
287                                     (minNode.myGame.getTitle().compareTo(current.myGame.getTitle())
288                                     > 0)//compare attributes
289                                     {
290                                             minNode = current;
291                                     }
292                             }
293                             //swap
294                             VideoGame temp = front.myGame;
295                             front.myGame = minNode.myGame;
296                             minNode.myGame = temp;
297                     }
298             }
299             if (sortInput.equals("p"))
300             {
301                     GameNode minNode;
302                     for (GameNode front = head; front != null; front = front.next)//traverse through linked list
303                     {
304                             minNode = front;
305                             for(GameNode current = front; current != null; current = current.next)//traverse
306                             through linked list
307                             {
308                                     if
309                                     (minNode.myGame.getPlatform().compareTo(current.myGame.getPlat
310                                     form()) > 0)//compare attributes
311                                     {
312                                             minNode = current;
313                                     }
314                             }
315                             //swap
316                             VideoGame temp = front.myGame;
317                             front.myGame = minNode.myGame;
318                             minNode.myGame = temp;
319                     }
320             }
321             if (sortInput.equals("pub"))
322             {
323                     GameNode minNode;
324                     for (GameNode front = head; front != null; front = front.next)//traverse through linked list
325                     {
326                             minNode = front;
327                             for(GameNode current = front; current != null; current = current.next)//traverse
328                             through linked list
329                             {
330                                     if
331                                     (minNode.myGame.getPublisher().compareTo(current.myGame.getPub
332                                     lisher()) > 0)//compare attributes
333                                     {
334                                             minNode = current;
335                                     }
336                             }
```

```
337                         //swap
338                         VideoGame temp = front.myGame;
339                         front.myGame = minNode.myGame;
340                         minNode.myGame = temp;
341                 }
342         }
343         if (sortInput.equals("d"))
344         {
345                 GameNode minNode;
346                 for (GameNode front = head; front != null; front = front.next)//traverse through linked list
347                 {
348                         minNode = front;
349                         for(GameNode current = front; current != null; current = current.next)//traverse
350                         through linked list
351                         {
352                                 if
353                                 (minNode.myGame.getDeveloper().compareTo(current.myGame.getDe
354                                 veloper()) > 0)//compare attributes
355                                 {
356                                         minNode = current;
357                                 }
358                         }
359                         //swap
360                         VideoGame temp = front.myGame;
361                         front.myGame = minNode.myGame;
362                         minNode.myGame = temp;
363                 }
364         }
365         if (sortInput.equals("g"))
366         {
367                 GameNode minNode;
368                 for (GameNode front = head; front != null; front = front.next)//traverse through linked list
369                 {
370                         minNode = front;
371                         for(GameNode current = front; current != null; current = current.next)//traverse
372                         through linked list
373                         {
374                                 if
375                                 (minNode.myGame.getGenre().compareTo(current.myGame.getGenre(
376                                 )) > 0)//compare attributes
377                                 {
378                                         minNode = current;
379                                 }
380                         }
381                         //swap
382                         VideoGame temp = front.myGame;
383                         front.myGame = minNode.myGame;
384                         minNode.myGame = temp;
385                 }
386         }
387         if (sortInput.equals("r"))
388         {
389                 GameNode minNode;
390                 for (GameNode front = head; front != null; front = front.next)//traverse through linked list
391                 {
392                         minNode = front;
```

```
393                                 for(GameNode current = front; current != null; current = current.next)//traverse
394                                 through linked list
395                                 {
396                                         if
397                                         (minNode.myGame.getRelease().compareTo(current.myGame.getRele
398                                         ase()) > 0)//compare attributes
399                                         {
400                                                 minNode = current;
401                                         }
402                                 }
403                                 //swap
404                                 VideoGame temp = front.myGame;
405                                 front.myGame = minNode.myGame;
406                                 minNode.myGame = temp;
407                         }
408                 }
409                 if (sortInput.equals("e"))
410                 {
411                         GameNode minNode;
412                         for (GameNode front = head; front != null; front = front.next)//traverse through linked list
413                         {
414                                 minNode = front;
415                                 for(GameNode current = front; current != null; current = current.next)//traverse
416                                 through linked list
417                                 {
418                                         if
419                                         (minNode.myGame.getESRB().compareTo(current.myGame.getESRB(
420                                         )) > 0)//compare attributes
421                                         {
422                                                 minNode = current;
423                                         }
424                                 }
425                                 //swap
426                                 VideoGame temp = front.myGame;
427                                 front.myGame = minNode.myGame;
428                                 minNode.myGame = temp;
429                         }
430
431                  }//if input is invalid, return error
432                  if (!sortInput.equals("id") && !sortInput.equals("t") && !sortInput.equals("p") &&
433                 !sortInput.equals("pub") && !sortInput.equals("d") && !sortInput.equals("g") &&
434                 !sortInput.equals("r") && !sortInput.equals("e"))
435                                         {
436                                                 System.out.println("\nSorry "' + sortInput + "' is a invalid
437                                                 selection");
438                                                 System.out.println("Please try again\n");
439                                         }
440         }
441         // idSearch(int idNum) traverses through the linked list looking for Nodes with the same ID attributed equal
442         to the int idNum input by the user
443         public int idSearch(int idNum)
444         {
445                 int count = 0;
446                 for (GameNode temp = head; temp!= null; temp = temp.next)//traverse through linked list
447                 {
448                         if (Integer.toString(temp.myGame.getID()).equals(Integer.toString(idNum)))
```

```
449                             {//compare IDs, if equal return index
450                                     return count;
451                             }
452                             count++;
453                     }
454                     return -1;
455             }
456             //reNum() traverses through the linked list renumbering all ID attributes of all nodes
457             public void reNum()
458             {
459                     int id = 0;//initialize int id
460                     for (GameNode temp = head; temp!= null; temp = temp.next)//traverse through linked list
461                     {
462                             temp.myGame.setID(id);//set ID equal to int id
463                             id++;//increase value of id
464                     }
465             }
```

```
1   /* Project Name: IB Computer Science Dossier
2    * Program Name: Video Game Database Creator/Editor
3    * File Name: VGDB.java
4    * Author: Sean Michael Hoffman
5    * Date: March 23, 2010
6    * School: Washington-Lee High School
7    * IDE: JCreator
8    */
9   public class VGDB
10  {
11          private DB myGames;
12
13          public VGDB()
14          {
15                  myGames = new DB();//initialize new database
16          }
17          public VGDB(DB newGames)
18          {
19                  myGames = newGames;
20          }
21          public DB getMyGames()
22          {
23                  return myGames;
24          }
25          public void addGames(VideoGame newGame)//compile add method
26          {
27                  myGames.add(newGame);
28          }
29          public void removeGame (int count)//compile remove method
30          {
31                  myGames.remove(count);
32          }
33          public void reNum()//compile reNum method
34          {
35                  myGames.reNum();
36          }
37          public int size()//return size
38          {
39                  return myGames.size();
40          }
41          public VideoGame find(String title, String platform)//compile find method
42          {
43                  int result = myGames.find(title, platform);
44                  //check if found
45                  if (result != -1)
46                  {
47                          return myGames.getGame(result);
48                  }
49                  return null;
50          }
51          public int idSearch(int idNum)//compile idSearch method
52          {
53                  return myGames.idSearch(idNum);
54          }
55          public void print()//compile print method
```

```
56              {
57                      myGames.print();
58              }
59              public void dbWrite(String fileName)//compile dbWrite method
60              {
61                      myGames.dbWrite(fileName);
62              }
63              public void alphaSort(String sortInput)//compile alphaSort method
64              {
65                      myGames.alphaSort(sortInput);
66              }
67              public VideoGame getGame(int index)//compile getGame method
68              {
69                      return myGames.getGame(index);
70              }
71      }
```

```
1   /* Project Name: IB Computer Science Dossier
2    * Program Name: Video Game Database Creator/Editor
3    * File Name: VGDBMain.java
4    * Author: Sean Michael Hoffman
5    * Date: March 23, 2010
6    * School: Washington-Lee High School
7    * IDE: JCreator
8    */
9   import java.io.*;
10
11  public class VGDBMain
12  {
13          public static void main ( String [] args ) throws IOException
14          {
15                  //initialize VGDB myGames
16                  VGDB myGames = new VGDB();
17
18                  int id = 0; //ID starts at zero
19                  for (;;){
20                          //Editor Menu
21                          System.out.println ("\nWelcome to video game database creator/editor \n");
22
23                          System.out.println ("(a)dd to database");
24                          System.out.println ("(p)rint database");
25                          System.out.println ("(e)dit video game information");
26                          System.out.println ("(r)emove video game from database");
27                          System.out.println ("(s)earch database");
28                          System.out.println ("(sort) games");
29                          System.out.println ("(save) database data");
30                          System.out.println ("(load) database data");
31                          System.out.println ("(q)uit \n");
32
33                          String input = getInput.getString("What would you like to do? (lowercase only)");
34
35                          if (input.equals("a")) //User has selected add
36                          {
37
38                                  System.out.println ("\nUSER SELECTED ADD");
39
40                                  int vgID = id;
41                                  id++;
42
43                                  String vgTitle = getInput.getString("\nPlease input the title"); //Input for title
44
45                                  String vgPlatform = getInput.getString("\nPlease input the platform"); //Input
46                                  for platform
47
48                                  String vgPublisher = getInput.getString("\nPlease input the publisher"); //Input
49                                  for publisher
50
51                                  String vgDeveloper = getInput.getString("\nPlease input the developer");//Input
52                                  for developer
53
54                                  String vgGenre = getInput.getString("\nPlease input the genre"); //Input for
55                                  genre
56
```

```
57                      String vgRelease = getInput.getRD("\nPlease input the release (Format:
58                      MM.DD.YYYY)"); //Input for release date
59
60                      String vgESRB = getInput.getString("\nPlease input the ESRB Rating"); //Input
61                      for ESRB rating
62                      // Compile all varibles together and add to database linked list
63                      myGames.addGames(new VideoGame (vgID, vgTitle, vgPlatform, vgPublisher,
64                      vgDeveloper, vgGenre, vgRelease, vgESRB));
65              }
66              if (input.equals("p")) //User has selected print
67              {
68                      System.out.println ("\nUSER SELECTED PRINT");
69                      if (myGames.size() == 0) //If linked list is empty, return error
70                      {
71                              System.out.println("\nSorry the database is empty, please add a
72                              videogame or load a saved database file");
73                      }
74                      else //Linked list is not empty
75                      {
76                              myGames.print();//Print out linked list
77                      }
78              }
79              if (input.equals("e")) //User selected edit
80              {
81                      if (myGames.size() == 0) //If linked list is empty, return error
82                      {
83                              System.out.println("\nSorry the database is empty, please add a
84                              videogame or load a saved database file");
85                      }
86                      else //If linked list is not empty, run edit
87                      {
88                              System.out.println ("\nUSER SELECTED EDIT");
89                              //User prompted for ID number
90                              String num = getInput.getString("\nPlease input the ID number of the
91                              Video Game you would like to edit");
92                              int idNum = Integer.parseInt(num); //input is parsed and id number is
93                              converted to an int
94                              int count = myGames.idSearch(idNum); //idNum is run through the
95                              idSearch method, index is returned as count
96
97                              System.out.println("\n" + myGames.getGame(count));//count is passed
98                              through the getGame method and game node is displayed
99                              //User is asked if they would like to edit this game node
100                             String editInput = getInput.getString("\nIs this the entry you would like
101                             edit? (y)/(n)? (lowercase only)");
102
103                             if (editInput.equals("y"))//User selected yes
104                             {
105                                     System.out.println ("\nEdit Options:\n");
106
107                                     System.out.println ("(t)itle");
108                                     System.out.println ("(p)latform");
109                                     System.out.println ("(pub)lisher");
110                                     System.out.println ("(d)evloper");
111                                     System.out.println ("(g)enre");
112                                     System.out.println ("(r)elease");
```

```
113                 System.out.println ("(e)srb");
114                 System.out.println ("(a)ll \n");
115
116                 String editInput2 = getInput.getString("Which attribute would
117                 you like to edit? (lowercase only)");
118                 //If input doesn't equal any of the options, return error
119                 if (!editInput2.equals("t") && !editInput2.equals("p") &&
120                 !editInput2.equals("pub") && !editInput2.equals("g") &&
121                 !editInput2.equals("r") && !editInput2.equals("e") &&
122                 !editInput2.equals("a"))
123                 {
124                         System.out.println("\nSorry '" + editInput2 + "' is a
125                 invalid selection");
126                         System.out.println("Please try again\n");
127                         continue;
128                 }
129                 //if statements for this edit menu
130                         //Current attribute is displayed
131                         //User is prompted for input
132                         //input is set to attribute
133                 if (editInput2.equals("t"))
134                 {
135                         System.out.println("\nCurrent Title: " +
136                         myGames.getGame(count).getTitle());
137                         String eTitle = getInput.getNPString();
138                         myGames.getGame(count).setTitle(eTitle);
139                 }
140                 if (editInput2.equals("p"))
141                 {
142                         System.out.println("\nCurrent Platform: " +
143                         myGames.getGame(count).getPlatform());
144                         String ePlatform = getInput.getNPString();
145                         myGames.getGame(count).setPlatform(ePlatform);
146                 }
147                 if (editInput2.equals("pub"))
148                 {
149                         System.out.println("\nCurrent Publisher: " +
150                         myGames.getGame(count).getPublisher());
151                         String ePublisher = getInput.getNPString();
152                         myGames.getGame(count).setPublisher(ePublisher);
153                 }
154                 if (editInput2.equals("d"))
155                 {
156                         System.out.println("\nCurrent Developer: " +
157                         myGames.getGame(count).getDeveloper());
158                         String eDeveloper = getInput.getNPString();
159
160                         myGames.getGame(count).setDeveloper(eDeveloper
161                 );
162                 }
163                 if (editInput2.equals("g"))
164                 {
165                         System.out.println("\nCurrent Genre: " +
166                         myGames.getGame(count).getGenre());
167                         String eGenre = getInput.getNPString();
168                         myGames.getGame(count).setGenre(eGenre);
```

```
169                                     }
170                                     if (editInput2.equals("r"))
171                                     {
172                                             System.out.println("\nCurrent Release: " +
173                                             myGames.getGame(count).getRelease());
174                                             String eRelease = getInput.getNPRD();
175                                             myGames.getGame(count).setRelease(eRelease);
176                                     }
177                                     if (editInput2.equals("e"))
178                                     {
179                                             System.out.println("\nCurrent ESRB: " +
180                                             myGames.getGame(count).getESRB());
181                                             String eESRB = getInput.getNPString();
182                                             myGames.getGame(count).setESRB(eESRB);
183                                     }
184                                     if (editInput2.equals("a")) //all attributes are selected
185                                     {
186                                             System.out.println("\nCurrent Title: " +
187                                             myGames.getGame(count).getTitle());
188                                             String eTitle = getInput.getNPString();
189                                             myGames.getGame(count).setTitle(eTitle);
190
191                                             System.out.println("\nCurrent Platform: " +
192                                             myGames.getGame(count).getPlatform());
193                                             String ePlatform = getInput.getNPString();
194                                             myGames.getGame(count).setPlatform(ePlatform);
195
196                                             System.out.println("\nCurrent Publisher: " +
197                                             myGames.getGame(count).getPublisher());
198                                             String ePublisher = getInput.getNPString();
199                                             myGames.getGame(count).setPublisher(ePublisher);
200
201                                             System.out.println("\nCurrent Developer: " +
202                                             myGames.getGame(count).getDeveloper());
203                                             String eDeveloper = getInput.getNPString();
204
205                                             myGames.getGame(count).setDeveloper(eDeveloper
206                                     );
207
208                                             System.out.println("\nCurrent Genre: " +
209                                             myGames.getGame(count).getGenre());
210                                             String eGenre = getInput.getNPString();
211                                             myGames.getGame(count).setGenre(eGenre);
212
213                                             System.out.println("\nCurrent Release: " +
214                                             myGames.getGame(count).getRelease());
215                                             String eRelease = getInput.getNPRD();
216                                             myGames.getGame(count).setRelease(eRelease);
217
218                                             System.out.println("\nCurrent ESRB: " +
219                                             myGames.getGame(count).getESRB());
220                                             String eESRB = getInput.getNPString();
221                                             myGames.getGame(count).setESRB(eESRB);
222                                     }
223                                     if (!editInput2.equals("t") && !editInput2.equals("p") &&
```

```
224                                     !editInput2.equals("pub") && !editInput2.equals("d") &&
225                                     !editInput2.equals("g") && !editInput2.equals("r") &&
226                                     !editInput2.equals("e") && !editInput2.equals("a"))
227                                     {
228                                             System.out.println("\nSorry '" + editInput + "' is a
229                                             invalid selection");
230                                             System.out.println("Please try again\n");
231                                     }
232
233                             }
234                             if (editInput.equals("n"))//user selected no
235                             {
236                                     continue;
237                             }
238                             if (!editInput.equals("y") && !editInput.equals("n")) //user did not
239                             input yes or no
240                             {
241                                     System.out.println("\nSorry '" + editInput + "' is a invalid
242                                     selection");
243                                     System.out.println("Please try again\n");
244                             }
245                     }
246             }
247             if (input.equals("s"))
248             {
249
250                     System.out.println ("\nUSER SELECTED SEARCH");
251
252                     String title = getInput.getString("\nPlease input the title");
253
254                     String platform = getInput.getString("\nPlease input the platform");
255
256                     //inputs passed through find method
257                     VideoGame searchResult = myGames.find(title , platform);
258                     if (searchResult == null)//Game node is not found
259                     {
260                             System.out.print("\nVideo game not found\n");
261                     }else{//game node is found
262                             System.out.print("\nVideo game found!\n");
263                             System.out.print("[" + searchResult + "]\n");
264                     }
265             }
266             if (input.equals("sort"))
267             {
268                     if (myGames.size() == 0) //If linked list is empty, return error
269                     {
270                             System.out.println("\nSorry the database is empty, please add a
271                             videogame or load a saved database file");
272                     }
273                     else //if linked list is not empty, run sort
274                     {
275                             System.out.println ("USER SELECTED SORT");
276                             System.out.println ("How would you like to sort your contacts?\n");
277                             System.out.println ("(s)election sort");
278                             String input1 = getInput.getString("\nHow would you like to sort?");
279                             if (!input1.equals("s"))//input is not equal to selection sort
```

```
280                                     {
281                                             System.out.println("\nSorry "' + input1 + "' is a invalid
282                                             selection");
283                                             System.out.println("Please try again\n");
284                                             continue;
285                                     }
286                                     if (input1.equals("s"))
287                                     {       //display sort options
288                                             System.out.println ("\nSorting Options:\n");
289
290                                             System.out.println ("(id)");
291                                             System.out.println ("(t)itle");
292                                             System.out.println ("(p)latform");
293                                             System.out.println ("(pub)lisher");
294                                             System.out.println ("(d)evloper");
295                                             System.out.println ("(g)enre");
296                                             System.out.println ("(r)elease");
297                                             System.out.println ("(e)srb \n");
298
299                                             String sortInput = getInput.getString("Which attribute would
300                                             you like to sort alphabetically? (lowercase only)");
301
302                                             myGames.alphaSort(sortInput);//input is passed through
303                                             alphaSort method
304                                             System.out.print("\nSorted Linked list\n");
305                                             myGames.print();
306                                     }
307                             }
308                     }
309                     if (input.equals("save"))
310                     {
311                             if (myGames.size() == 0)//if linked list is empty return error
312                             {
313                                     System.out.println("\nSorry the database is empty, please add a
314                                     videogame or load a saved database file");
315                                     continue;
316                             }
317                             else //linked list is not empty
318                             {
319
320                                     System.out.println ("\nUSER SELECTED SAVE");
321
322                                     //prompt user for filename
323                                     System.out.println("\nNOTE: File will automatically be saved to
324                                     program directory");
325                                     String fileName = getInput.getString("\nPlease input file name (do not
326                                     include file extension)");
327                                     myGames.alphaSort("id");//sort by id
328                                     myGames.dbWrite(fileName);//save file
329
330                                     System.out.println ("\nSaving...");
331                                     System.out.println ("\nDone.\n");
332                     }
333                     }
334                     if (input.equals("load"))
335                     {
```

```
336                     System.out.println ("\nUSER SELECTED LOAD");
337                     //prompt user for file name
338                     int listSize = myGames.size();
339                     if (listSize != 0)
340                     {
341                             for (int count = 0; count < listSize; count++)
342                             {
343                                     int t1 = myGames.idSearch(count);
344                                     myGames.removeGame(t1);
345                             }
346                     }
347                     System.out.println("\nNOTE: File will automatically be loaded from program
348                     directory");
349                     String fileName = getInput.getString("\nPlease input file name (do not include
350                     file extension)");
351                     try
352                     {
353                             RandomAccessFile file = new RandomAccessFile(fileName + ".txt",
354                             "r");
355                             String fileSize = file.readUTF();//read filesize (i.e. linked list size)
356                             int loopSize = Integer.parseInt(fileSize);//parse fileSize, convert to int
357
358                             for (int i = 0; i < loopSize; i++)//loop runs while i is less than loopsize
359                             {
360                                     int vgID = Integer.parseInt(file.readUTF());
361                                     String vgTitle = file.readUTF();
362                                     String vgPlatform = file.readUTF();
363                                     String vgPublisher = file.readUTF();
364                                     String vgDeveloper = file.readUTF();
365                                     String vgGenre = file.readUTF();
366                                     String vgRelease = file.readUTF();
367                                     String vgESRB = file.readUTF();
368                                     //compiles strings and adds game to database
369                                     myGames.addGames(new VideoGame (vgID, vgTitle,
370                                     vgPlatform, vgPublisher, vgDeveloper, vgGenre, vgRelease,
371                                     vgESRB));
372                             }
373                             file.close();//close file
374
375                             myGames.alphaSort("id");
376                             myGames.reNum();//renumber linked list
377                             System.out.println ("\nLoading...");
378                             System.out.println ("\nDone.\n");
379                     }
380                     catch (IOException e)//if file doesn't exist
381                     {
382                             System.out.println("\nERROR: File not Found. Try Again");
383                     }
384             }
385             if (input.equals("r"))
386             {
387                     if (myGames.size() == 0) //If linked list is empty, return error
388                     {
389                             System.out.println("\nSorry the database is empty, please add a
390                             videogame or load a saved database file");
391                     }
```

```
392                     else //If linked list is not empty, run remove
393                     {
394                             System.out.println ("\nUSER SELECTED REMOVE");
395                             String num = getInput.getString("\nPlease input the ID number of the
396                             Video Game you would like to remove");
397                             int idNum = Integer.parseInt(num);//num is parsed and converted into
398                             int
399                             int count = myGames.idSearch(idNum);//traverse linked list until index
400                             is
401                             returned
402
403
404                             System.out.println ("\n" + myGames.getGame(count));//count is passed
405                             through getGame method, game node is printed out
406                             String removeInput = getInput.getString("\nIs this the entry you would
407                             like Remove? (y)/(n)? (lowercase only)");
408
409                             if (removeInput.equals("y"))//user selects yes
410                             {
411                                     myGames.removeGame(count);//remove game
412                                     myGames.alphaSort("id");//sort database in sequential order
413                                     myGames.reNum();//renumber linked list
414                             }
415                             if (removeInput.equals("n"))//user selects no
416                             {
417                                     continue;
418                             }
419                             if (!removeInput.equals("y") && !removeInput.equals("n"))//input not
420                             equal to yes or no, return error
421                             {
422                                     System.out.println("\nSorry '" + input + "' is a invalid
423                                     selection");
424                                     System.out.println("Please try again\n");
425                             }
426                     }
427             }
428             if (input.equals("q"))
429             {
430                     String quitInput = getInput.getString("\nWould you like to save before you quit?
431                     (y)/(n)? (lowercase only)");
432                     if (quitInput.equals("y"))//user would like to save before quitting
433                     {
434                             if (myGames.size() == 0)//if linked list is empty, return error
435                             {
436                                     System.out.println("\nSorry the database is empty, please add
437                                     a videogame or load a saved database file");
438                                     continue;
439                             }
440                             else //linked list is not empty
441                             {
442
443                                     System.out.println ("\nUSER SELECTED SAVE");
444
445                                     System.out.println("\nNOTE: File will automatically be saved
446                                     to program directory");
447                                     String fileName = getInput.getString("\nPlease input file name
```

```
448                                                     (do not include file extension)");
449                                                     myGames.dbWrite(fileName);
450
451                                                     System.out.println ("\nSaving...");
452                                                     System.out.println ("\nDone.\n");
453
454                                                     System.out.println ("\nUSER SELECTED QUIT");
455                                                     break;
456                                             }
457                                     }
458                                     if (quitInput.equals("n"))//user selected no
459                                     {
460                                             System.out.println ("\nUSER SELECTED QUIT");
461                                             break;
462                                     }
463                                     if (!quitInput.equals("y") && !quitInput.equals("n"))//input is not equal to yes or
464                                     no, return error
465                                     {
466                                             System.out.println("\nSorry '" + input + "' is a invalid selection");
467                                             System.out.println("Please try again\n");
468                                     }
469                             }
470                             //input is not equal to any of the option, return error
471                             if (!input.equals("a") && !input.equals("e") && !input.equals("r") &&
472                             !input.equals("save") && !input.equals("load") && !input.equals("s") &&
473                             !input.equals("p") && !input.equals("sort") && !input.equals("q"))
474                             {
475                                     System.out.println("\nSorry '" + input + "' is a invalid selection");
476                                     System.out.println("Please try again\n");
477                             }
478                     }
479             }
480     }
```

C2: Usability

Usability features are summarized in the following table. The "Demonstrated" column gives the locations of evidence of each usability feature within the paper:

Feature	Summary	Demonstrated
Database Editor	A program which allows you to add and edit database entries for video games	See Figure 1 on page 56
Helpful Textual Menu	The editor provides a clean textual menu system to help the end user navigate through various editor functions	See Figure 1 on page 56
Ability to save	The editor prompts the user for a file name and saves the document as a text file.	See Figure 20 on page 63
Ability to load	The editor prompts the user for a file name and loads document.	See Figures 21, 22, 23 on page 64
Ability to search database entries	A function inside the editor that allows searching through database entries for matches to specific search criteria	See Figure 11 on page 61
Ability to sort database entries	The editor includes a function that sorts the entries by a specific criteria entered by the user	See Figures 12, 13, 14, 15, 16, 17, 18, 19 on page 61, 62
Explicit error messages	The editor provides error checking in the case that inputs are containing unrecognized data or other problems.	See Figure 24, 25, 26, 27 on page 66, 67

C3: Handling Errors

The handling of errors is summarized in the following table. The "Demonstrated" column gives the locations of evidence of each error is handled within the paper:

Error	Countermeasure	Demonstrated
Invalid Input	If the user inputs an incorrect menu option the editor will return an error message and run the menu	See page 52; Line 470 - 472
File not found	If the user inputs an incorrect file name the editor will return and error message and run the menu	See page 50; Line 380 - 383
Empty Input	If the user does not input any value into the editor when prompts and tries to continue, the editor will return and error message and run the prompt again.	See page 30; Line 12-35
Format Error	If the user inputs an incorrect a date in the wrong format the editor will return and error message and run the prompt again	See page 31, 37; Line 57 - 112
Type Error	If the user inputs letter(s) instead of an digits the editor will return and error message and run the prompt again	See page 31, 32; Line 57 - 112
Range Error	If the user inputs an incorrect a date, that is out of range, the editor will return and error message and run the prompt again	See page 31, 32; Line 57 - 112

C4: Success of Program

The success of program is summarized in the following table. The "Reference Screenshot" column gives the locations of evidence of each goal completed within the paper:

Program Goals	Reference Screenshots
Sort games alphabetically by attribute	See Figures 12, 13, 14, 15, 16, 17, 18, 19 on pages 61, 62
Editing database entries	See Figures 9, 10 on page 62
Saving and Loading a database	See Figures 20, 21, 22, 23 on page 64
Users should be able to add and remove database entries through a graphical or textual user interface	See Figures 1, 2, 3, 6, 7, 8 on pages 56, 57, 59
Display all database entries in sequential or sorted order	See Figures 4, 5 on page 58
Menu system should be simple and easy to navigate.	See Figures 1 on page 56
Error checking to prevent input errors	See Figures 24, 25, 26, 27 on page 66, 67
Output should be presented in a ordered easy to read fashion	See Figures 4, 5 on page 58

D1: Test Output

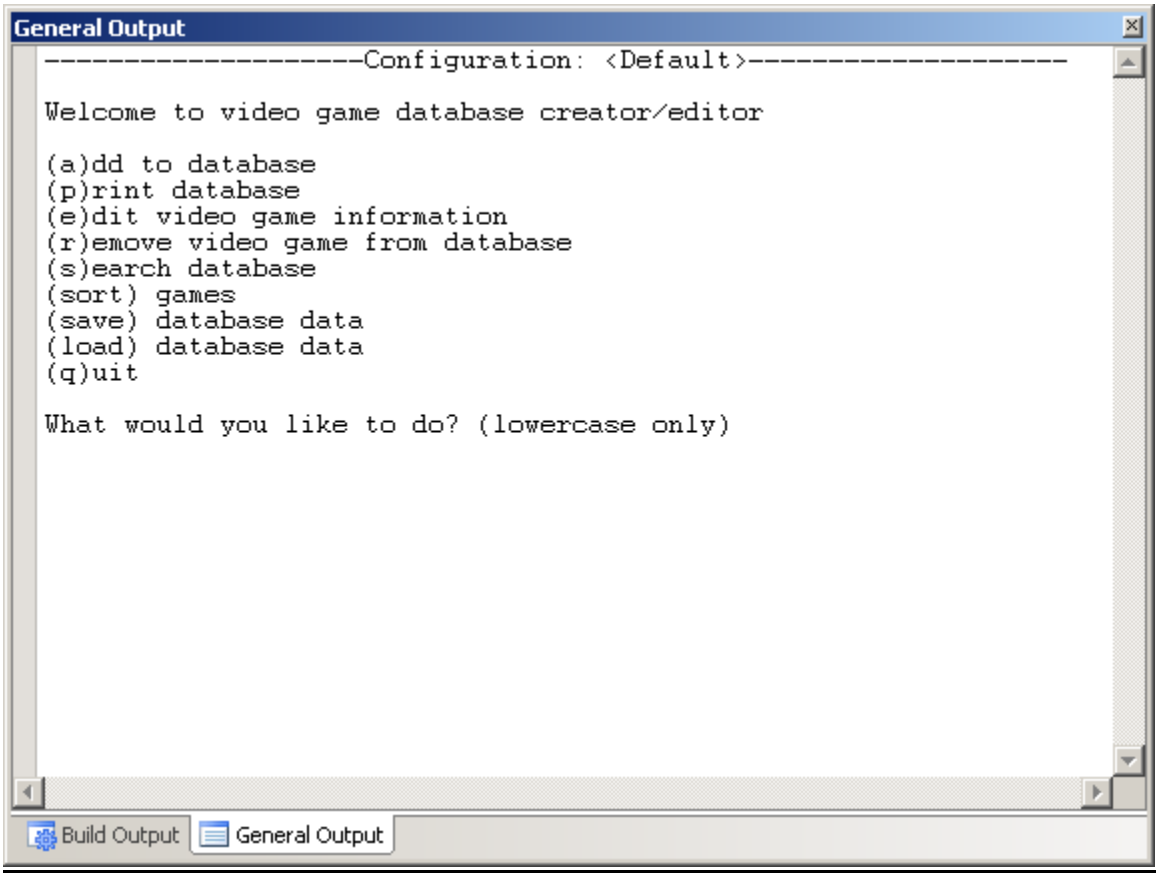


Figure 1 – Main Menu is displayed. The user is prompted for input from the list of options

Adding Entries to Database

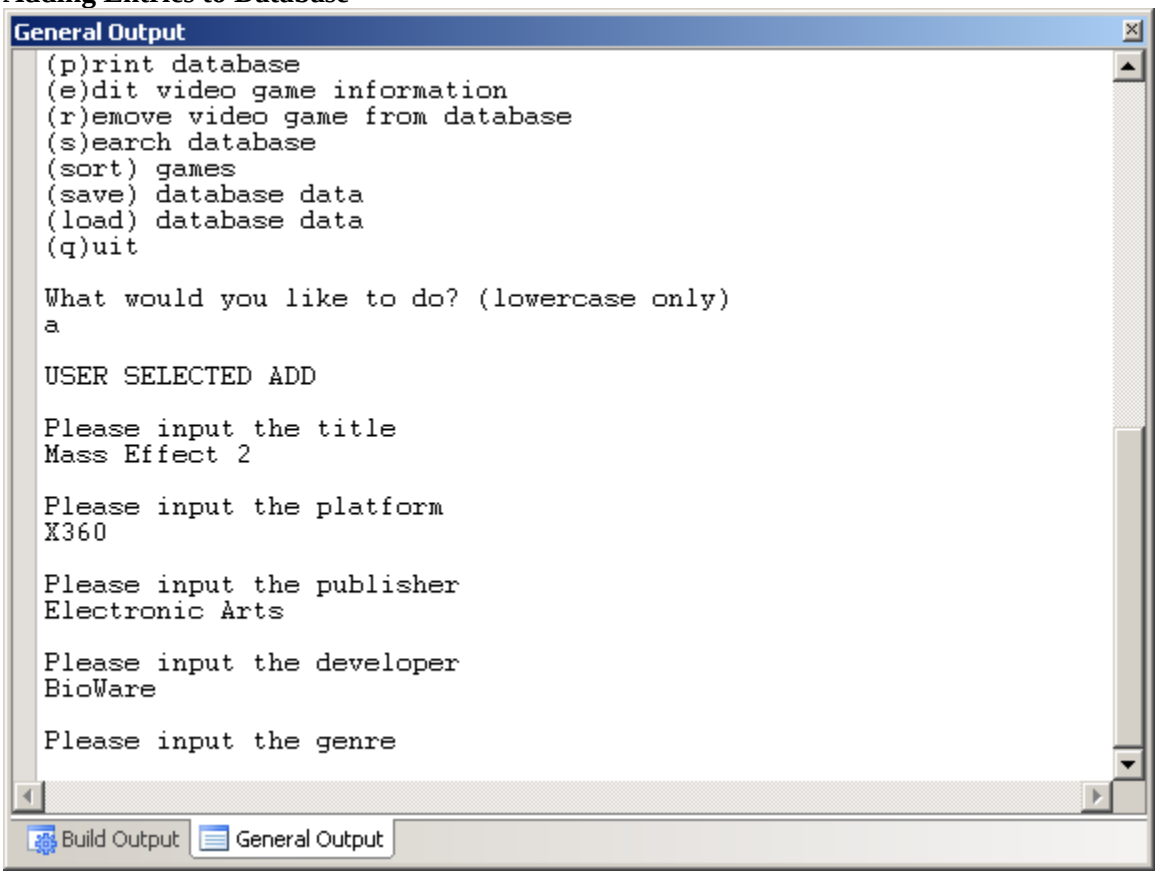


Figure 2 – Adding a video game to database. The user is prompted for the title, platform, publisher, developer, genre, release date, and esrb rating

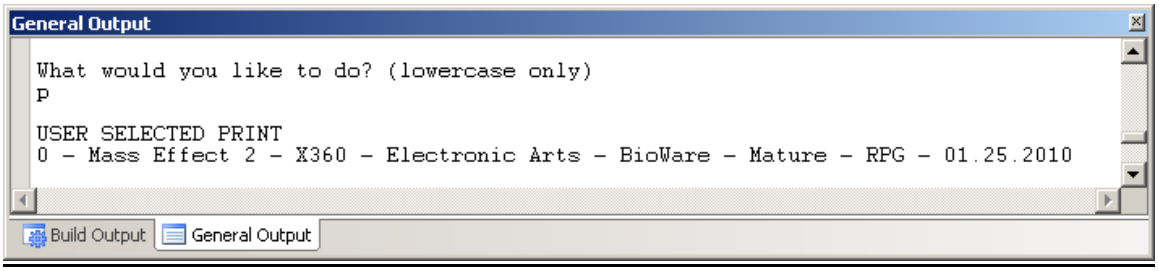


Figure 3 - Video game successfully added to database

Printing/Displaying Database

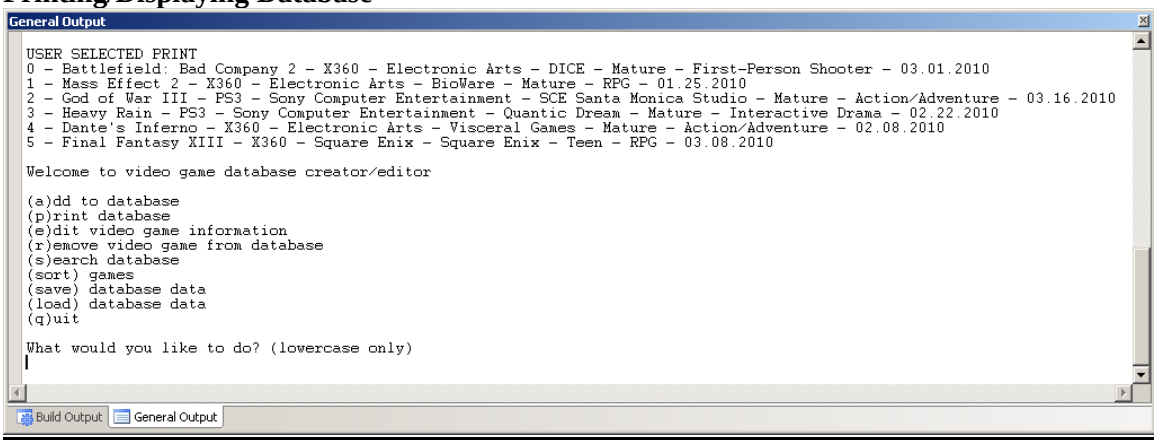


Figure 4 – Database is printed out in sequential order.

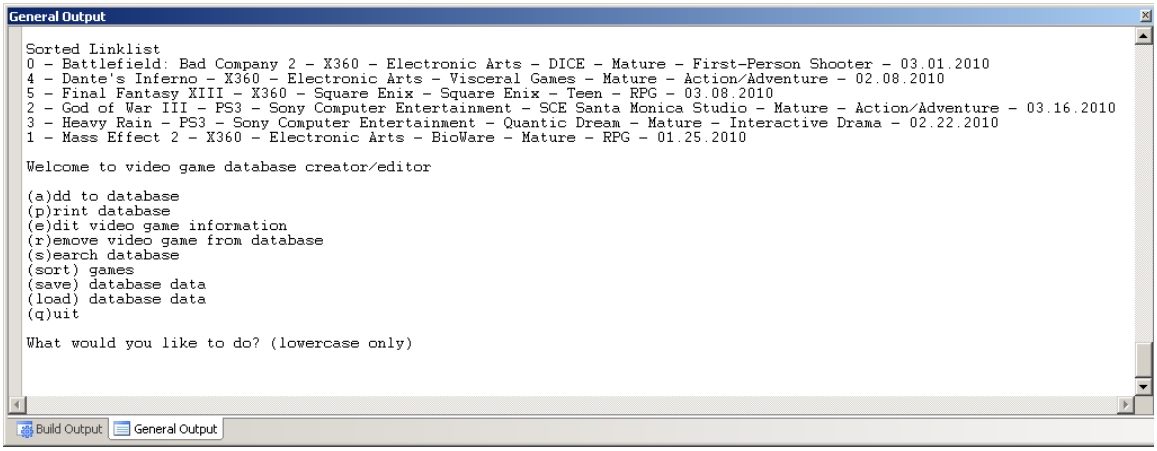


Figure 5 – Database printed out in alphabetical order by title.

Removing Entries from the Database

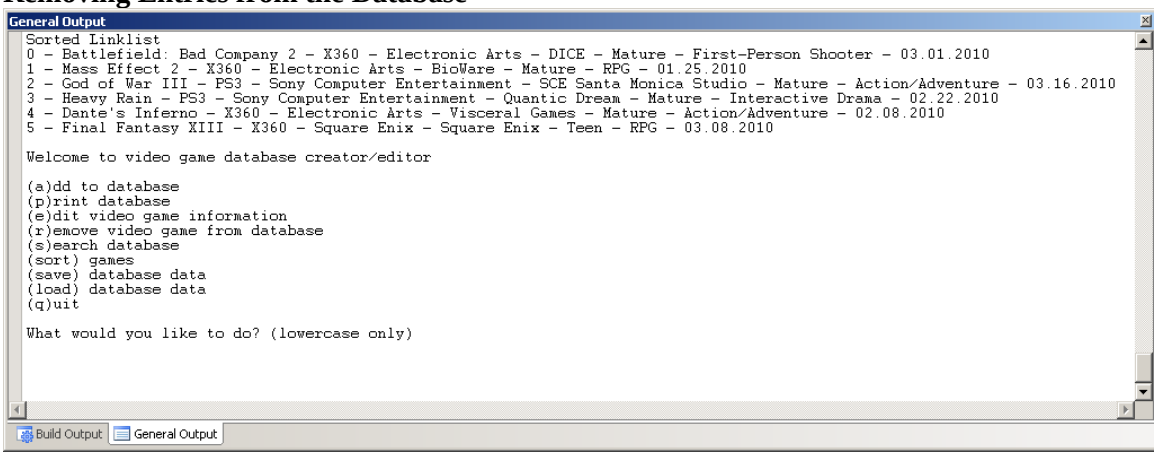


Figure 6 – The database before any entries are removed

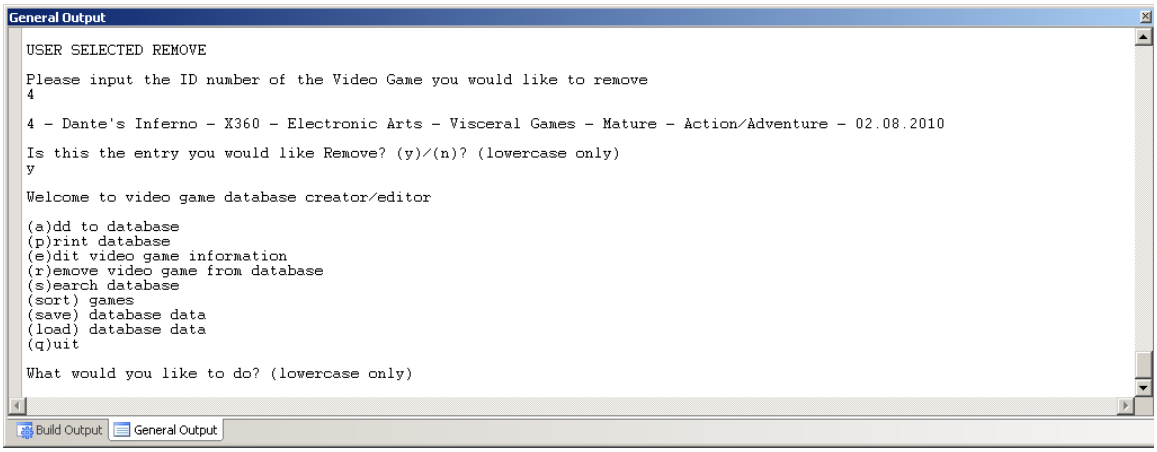


Figure 7 – Removing entry from database

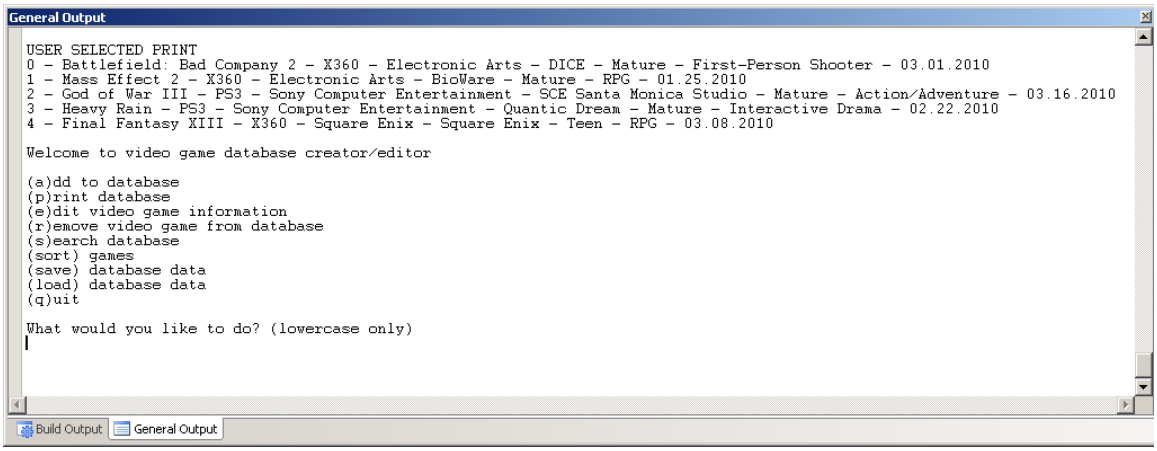


Figure 8 – Database after entry is removed

Editing Database Entries

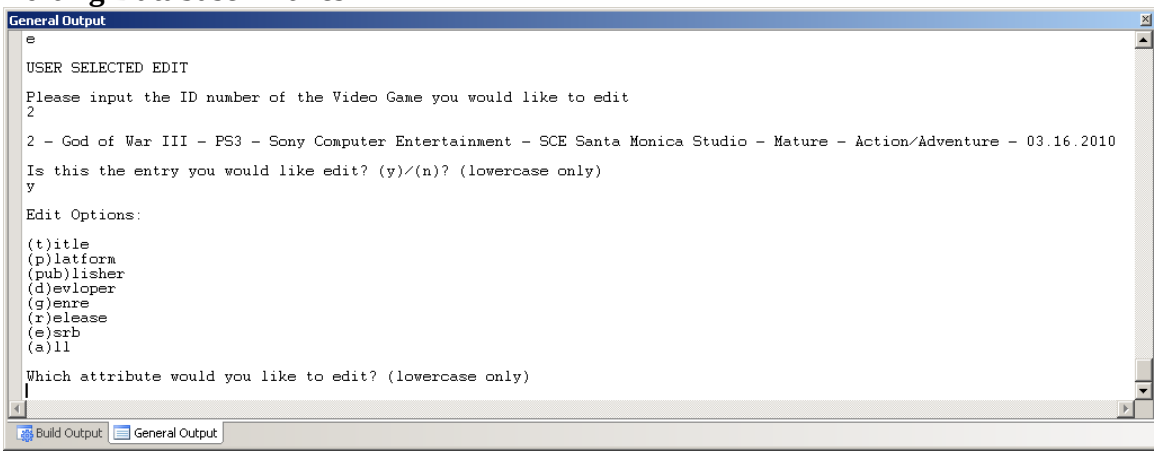


Figure 9 - Selecting entry to edit

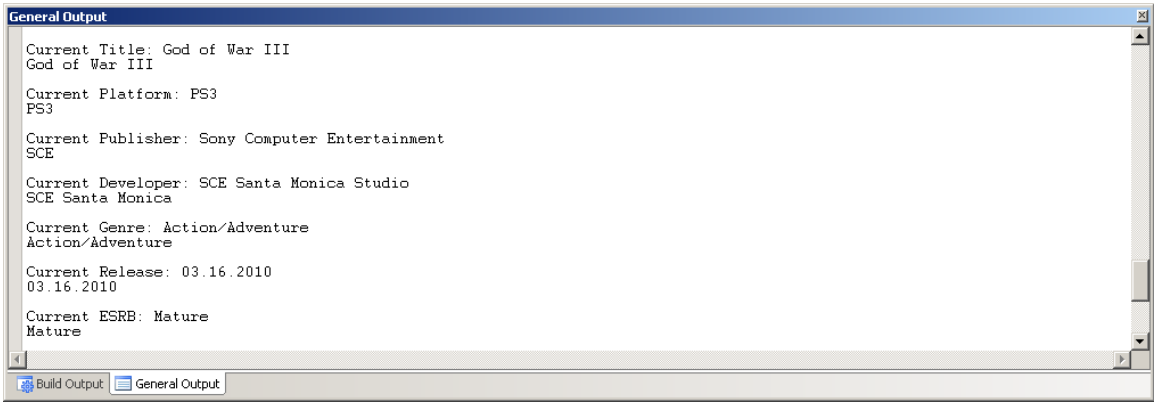


Figure 10 - Edit all attributes of entry

Searching for Database Entry

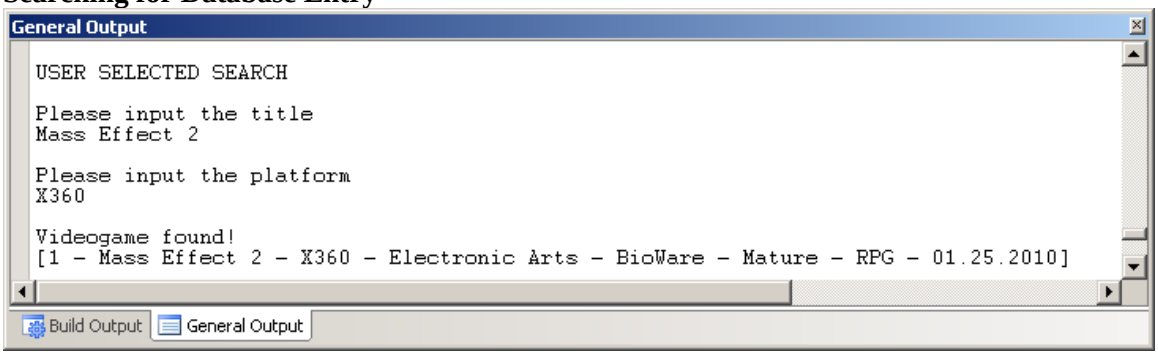


Figure 11 - Search function, searches through database and located the video game

Sorting the Database

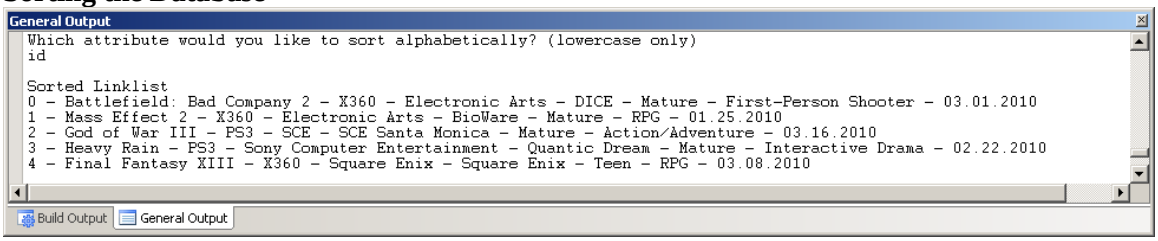


Figure 12 - Sorting database by ID

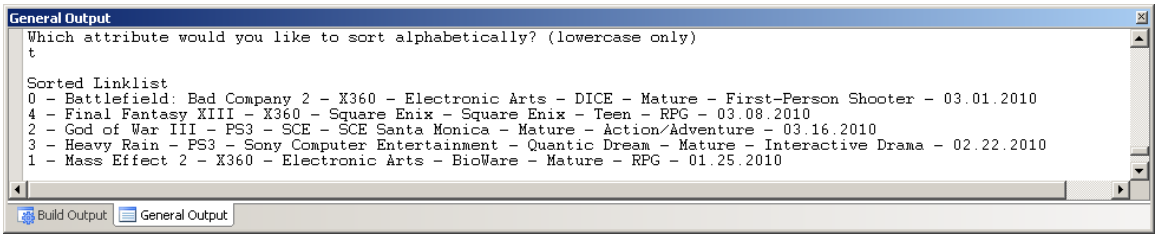


Figure 13 - Sorting database by title

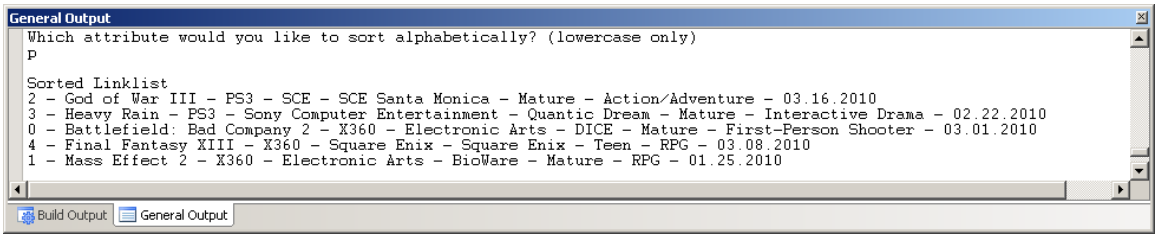


Figure 14 - Sorting database by platform

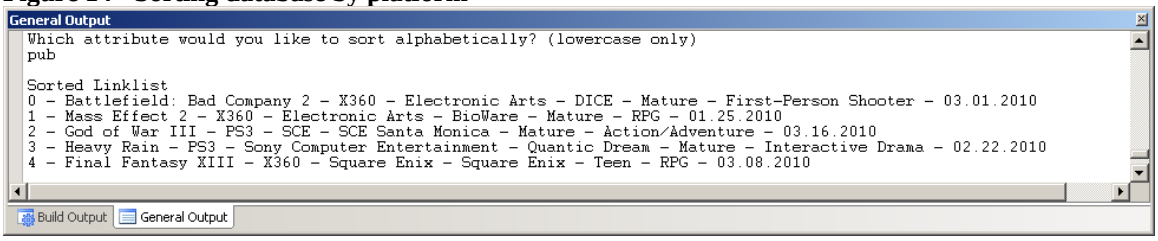


Figure 15 - Sorting database by publisher

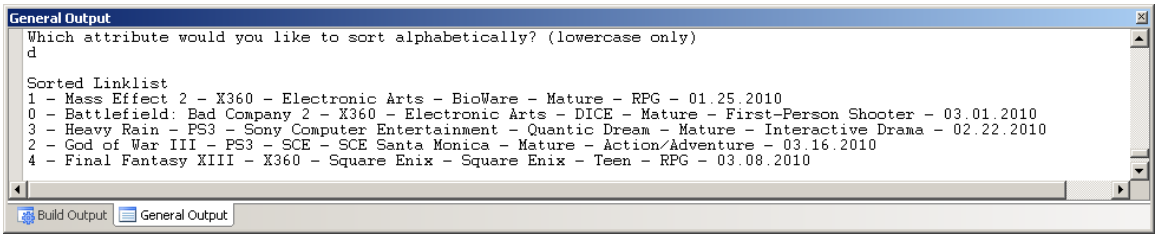


Figure 16 - Sorting database by developer

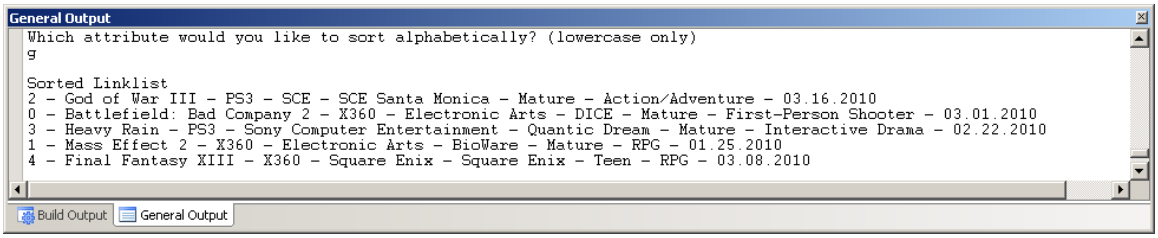


Figure 17 - Sorting database by genre

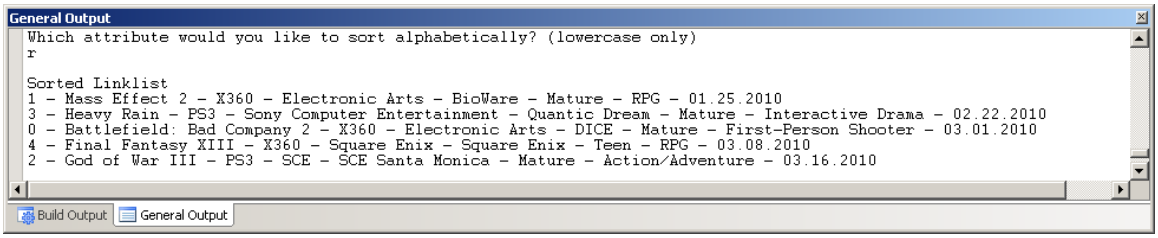


Figure 18 - Sorting database by release date

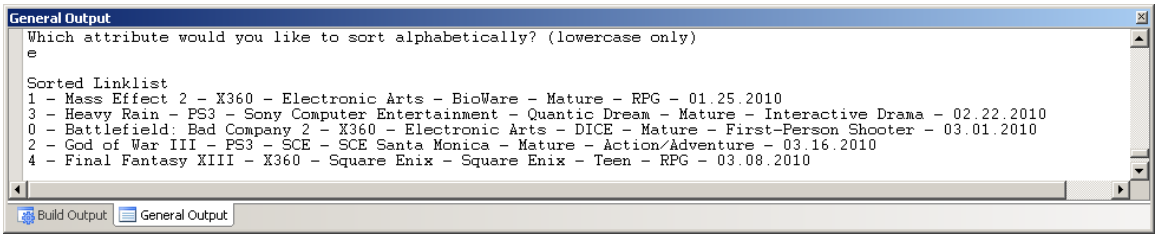


Figure 19 - Sorting database by ESRB

Saving Database

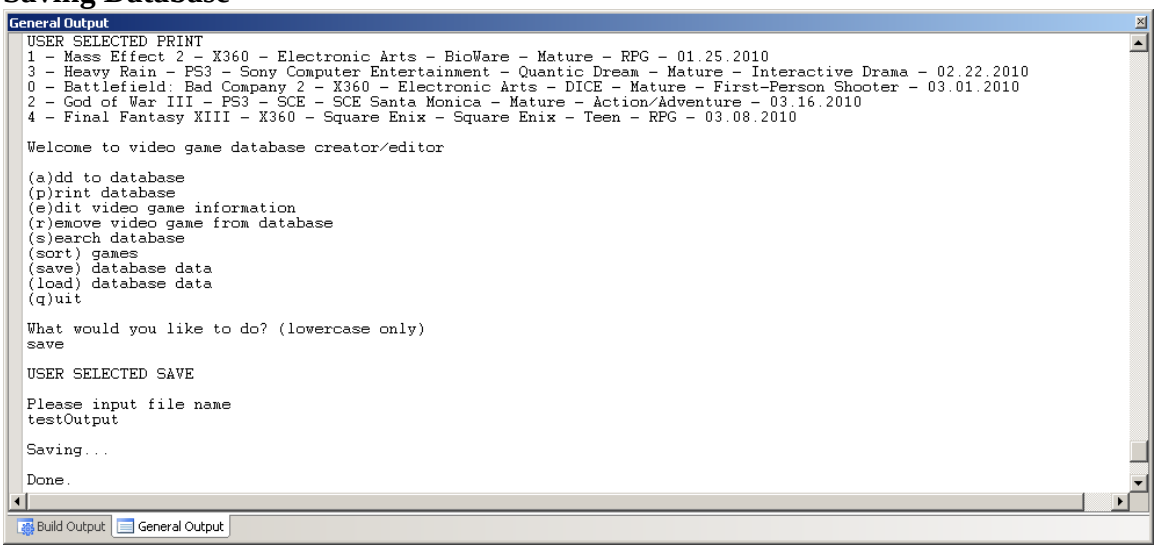


Figure 20 - Database is successfully saved

Loading Database

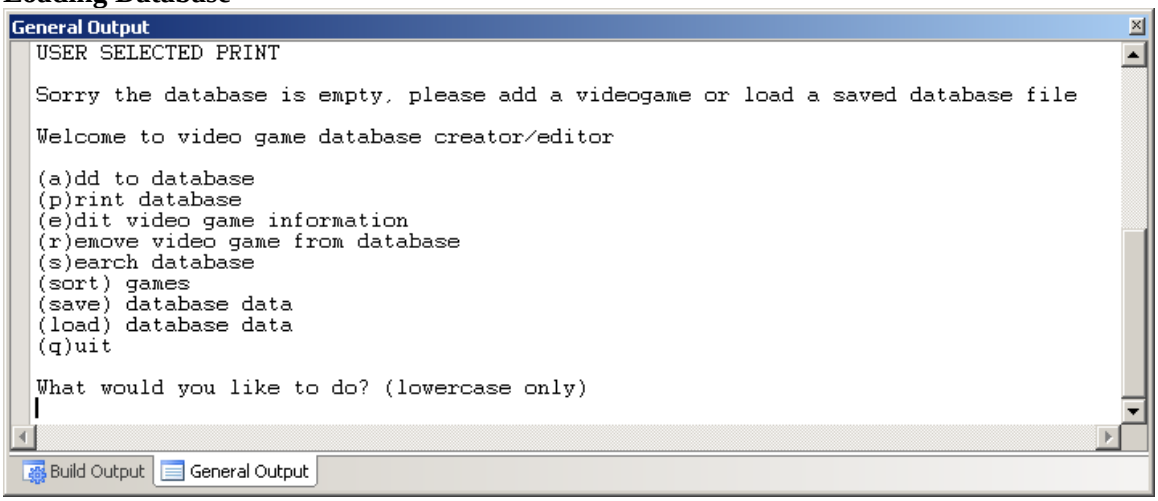


Figure 21 - Before loading database

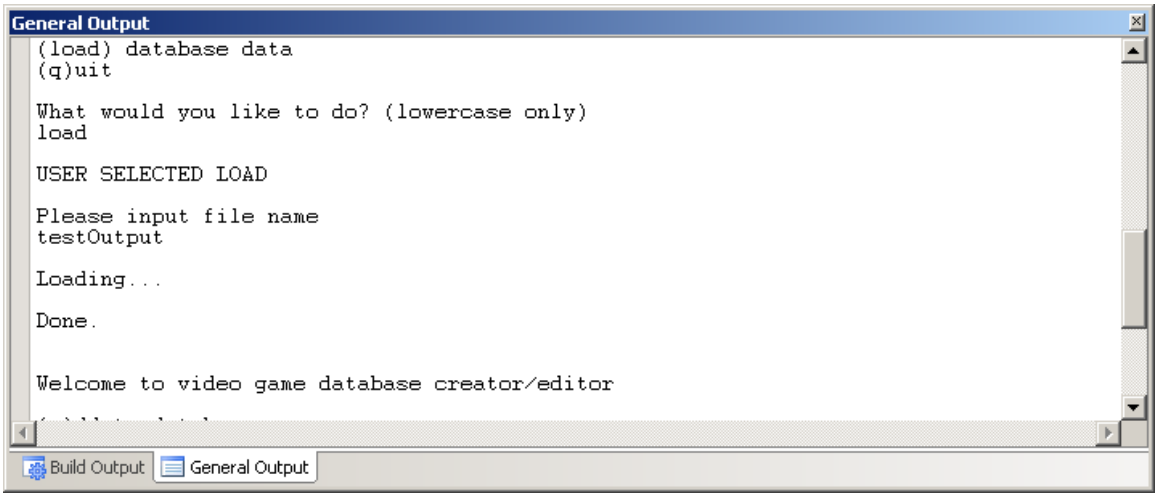


Figure 22 - File to load from is specified, file loaded successfully

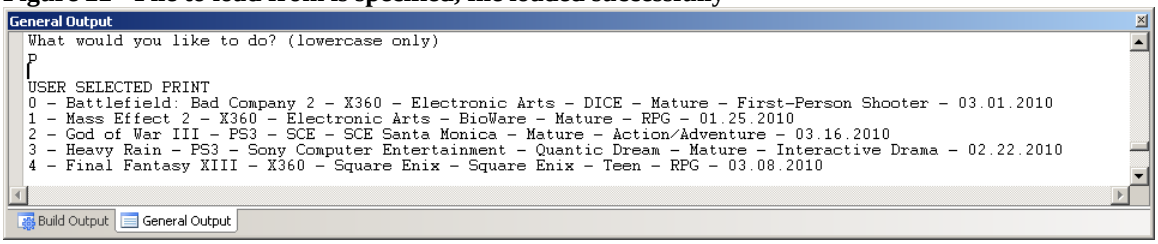


Figure 23 - Database after file has been loaded successfully

Quitting Database Editor

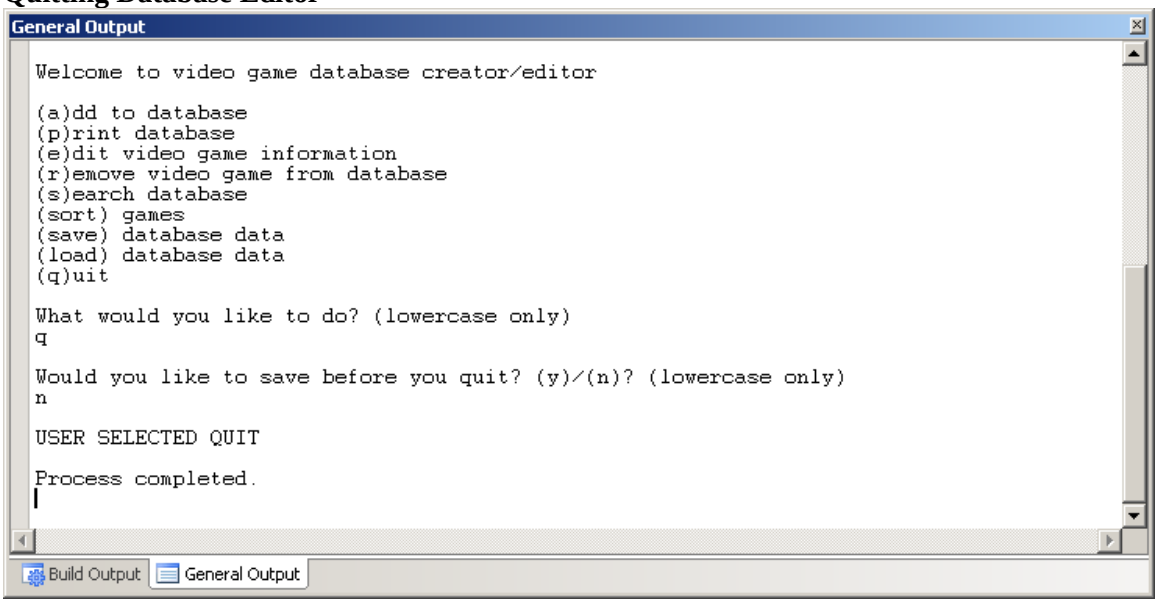


Figure 24 - User quits editor, Chooses not to save

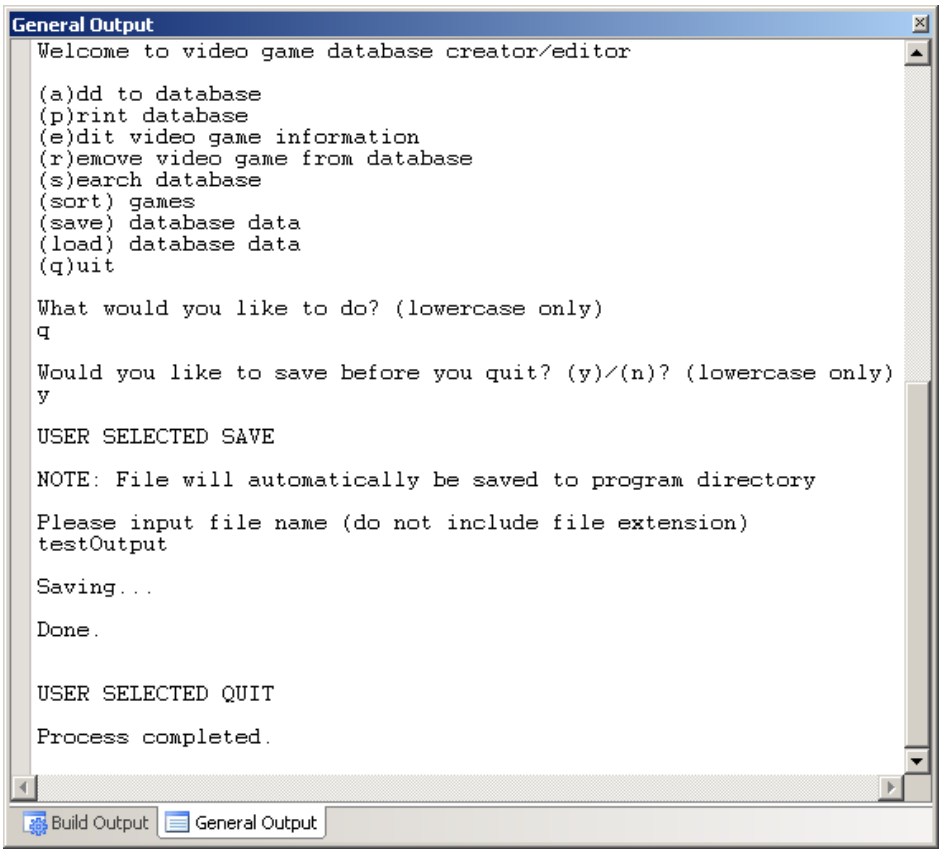


Figure 25 - User quits editor, Chooses to save

Error Handling

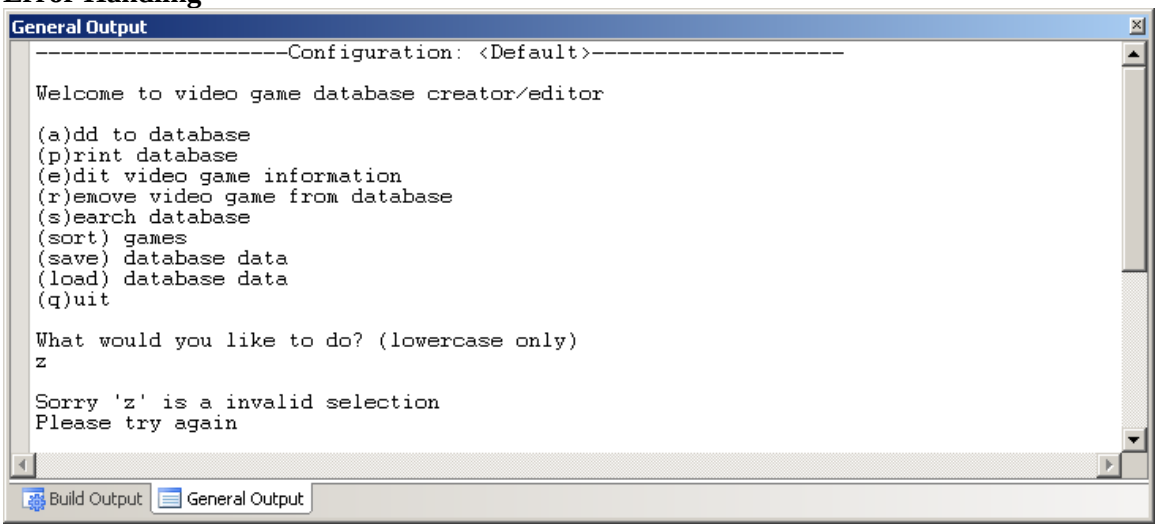


Figure 26 - Error Catching: Invalid Input

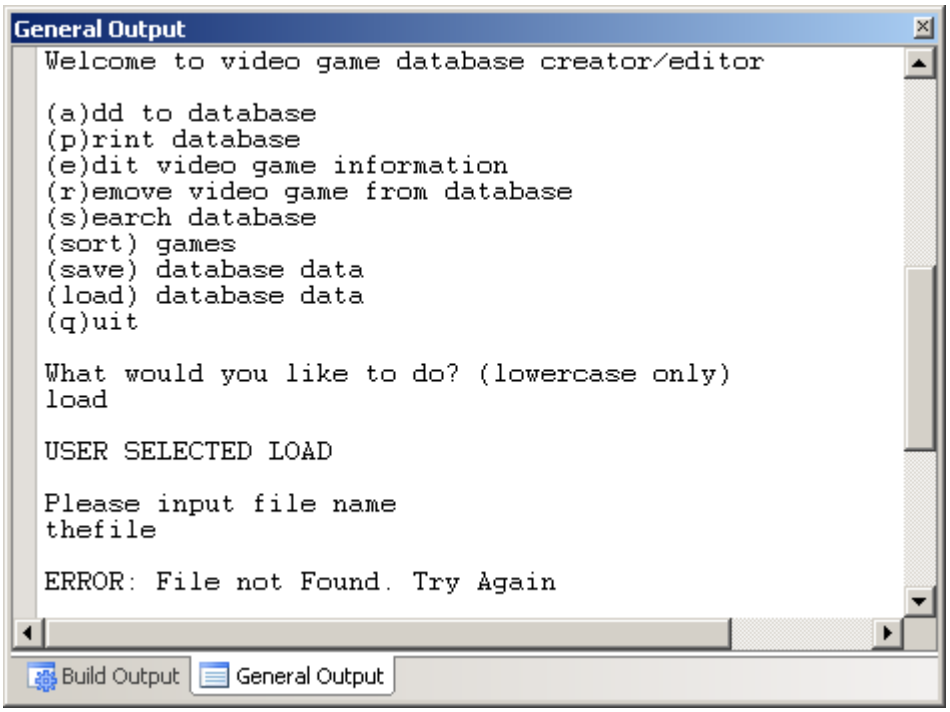


Figure 27 - Error Catching: File Not Found

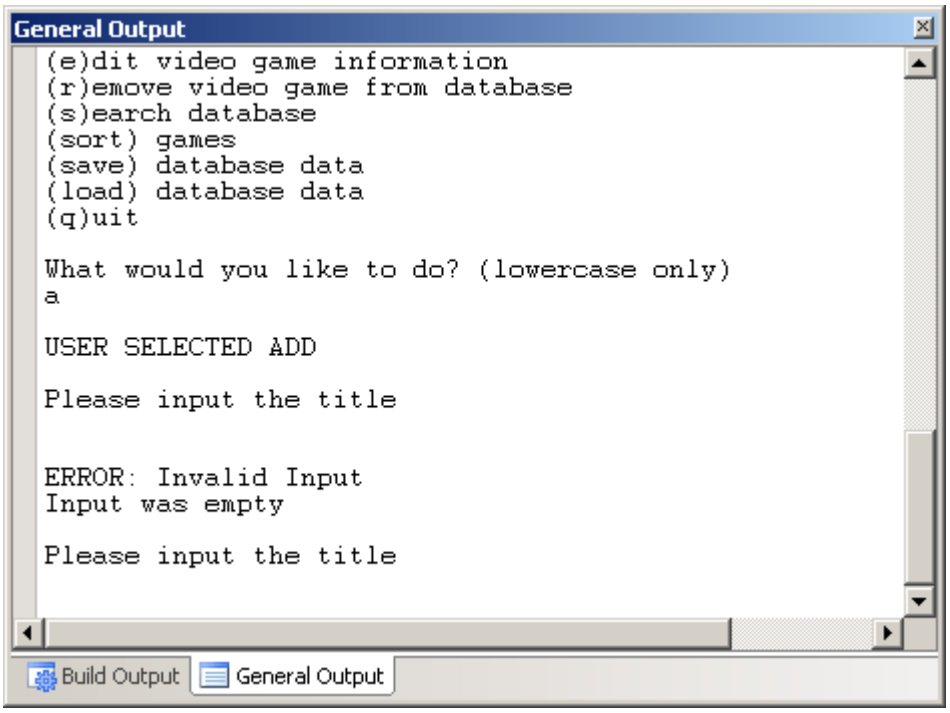


Figure 28 - Error Catching: Empty Input

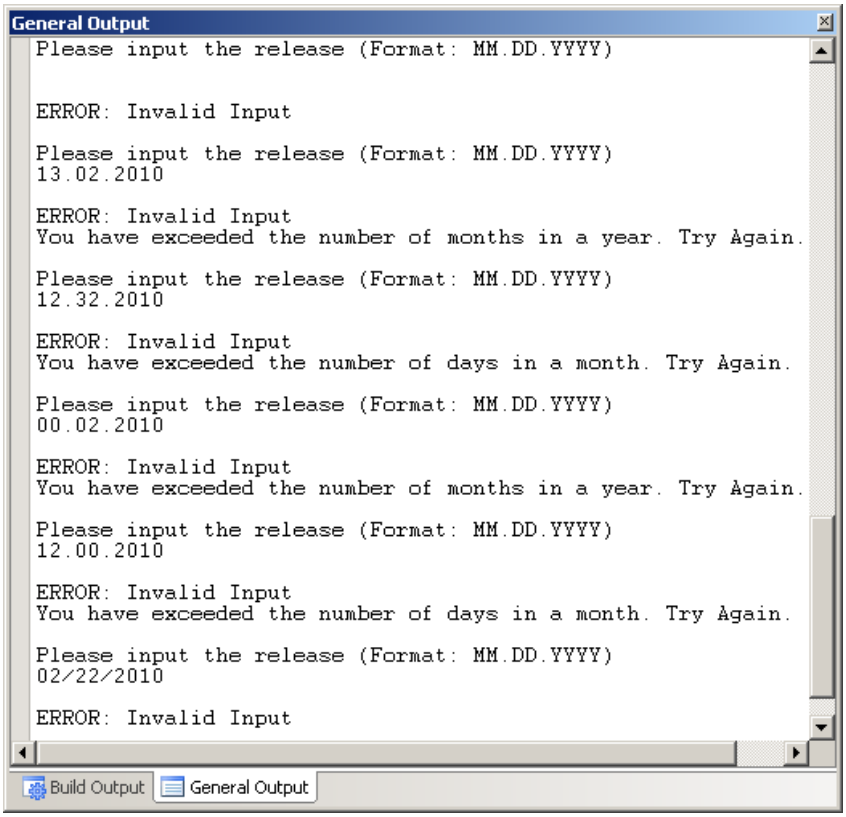


Figure 29 - Error Catching: Format Error, Invalid Types, Range Checking

D2: Conclusion

Success:
The program was a success and did met many of the requirements set in criteria for success. The goals that were outlined in criteria for success that were met include the following:

- Must be able to sort games alphabetically by the attributes stated above.
- Must allow end-user to edit specific information after the video game has already been added to the database.
- Ability to read/write database information from text files:\
- Should be able to detect any missing or corrupted information and inform the user.
- Users should be able to add, edit, remove database entries through a graphical or textual user interface
- Through the editor users should have the ability to name and save the database file as well as load databases
- Ability to sort video game criteria.
- Print a list of all database entries in sequential order
- Menu system should be simple and easy to navigate.
- Error checking to prevent input errors
- Output should be presented in a ordered easy to read fashion

Data Sets:
The program did not have any datasets that resulted in a error that crashed the program. Error-Checking and handling allow the program to catch certain data entry errors and re-prompt the user for better input.

Efficiency:
Generally the program works fast from the view of the end-user. However there are some improvements that could have been made to the system to allow the system to search and return data faster, such as a binary tree could have been used. A linear linked list search algorithm was used, which has a Big O complexity of O(n). For sorting, selection sort was used which has a Big O complexity of $O(n^2)$.

Limitations:
At this point the program is limited to seven basic information inputs. This limits the amount of information returned to the user when traversing the database. However this could easily be changed with an update to the program.

Additional Features:
To improve the speed of the program a binary tree could have been used rather than a linked list. The binary tree search function has a Big O complexity of O(log n), which is much faster than the linear search O(n). To make the database editor useful for larger game collections and administrative system would be added. Allowing users to login with usernames and passwords and retrieve information as well as requesting to borrow a game from the collection. The administrator would get exclusive privileges to add, edit, remove, and approve borrowing requests.

Appropriateness of Initial Design:
The initial design assisted in providing a architecture for the program. During the creation of the program appropriate modifications were made to the system that was beneficial to the end-product. The options that were made available during this process allowed the user to have more control over the system, without directly interfering with its structure. However the end-product design did not match up with the design of the initial prototype solution, but still kept the basic elements

Future Design and Enhancements:
If I had more extensive knowledge of the Java framework I would overhaul the system's interface and replace it with a graphical user interface. This would allow the program to be much more user-friendly as well as create a more streamlined interaction between the user and the system.

D3: User Documentation

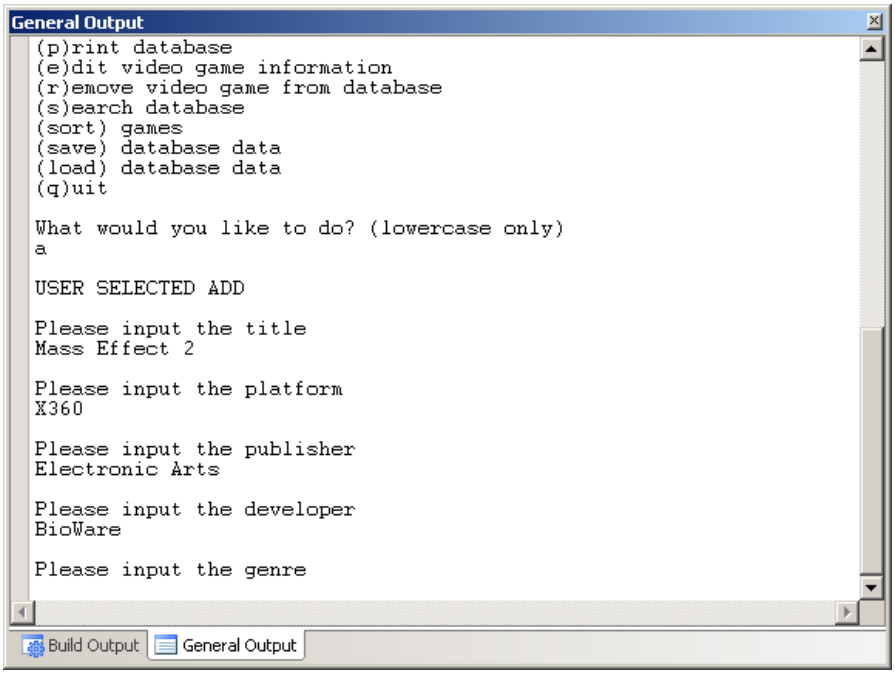
1. Installing and running the editor
    a. The first step to installing the editor is to download the following six java files into a folder on your computer: VideoGame.java, GameNode.java, DB.java, VGDB.java, VGDBMain.java, getInput.java
    b. Running the Editor
        i. If you are using Unix/Linux or Windows open of the command prompt/terminal and navigate to the folder where the files are located and compile the main file, VGDBMain.java, by executing the following command

            javac VGDBMain.java

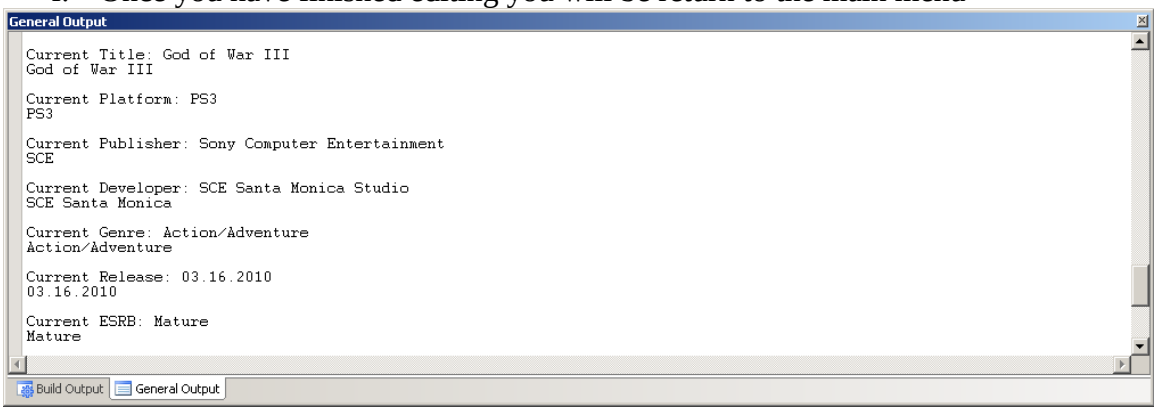
            After the main compiles successfully you can run the program by executing the following command:

            java VGDBMain

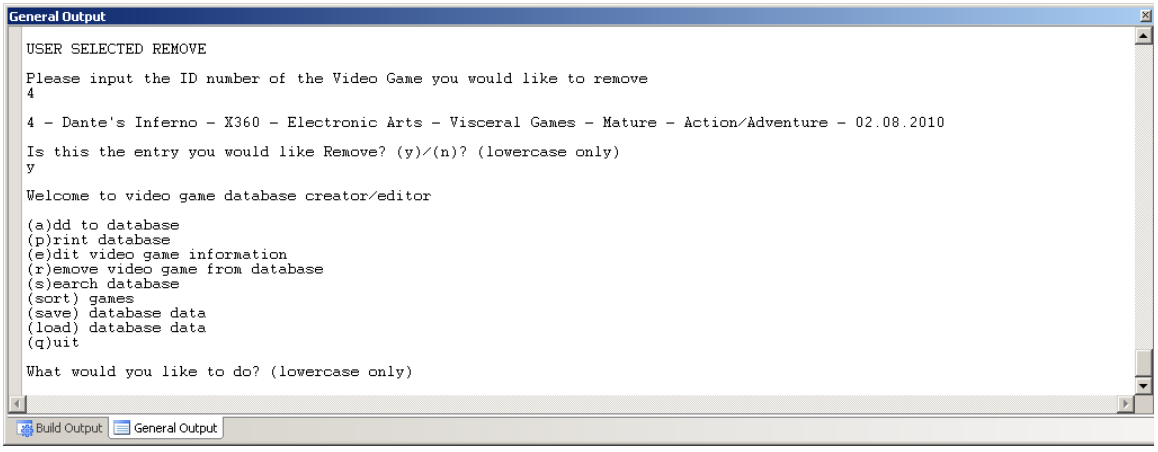
        ii. If are you are using a Java IDE, just simply open up the IDE and open VGDBMain.java and run the file.
2. Add a video game
    a. From the main menu type "a" and press "Enter"
    b. You will be prompted for the title, platform, publisher, developer, genre, release date, and ESRB rating. After completing the inputs the video game will be added to the database



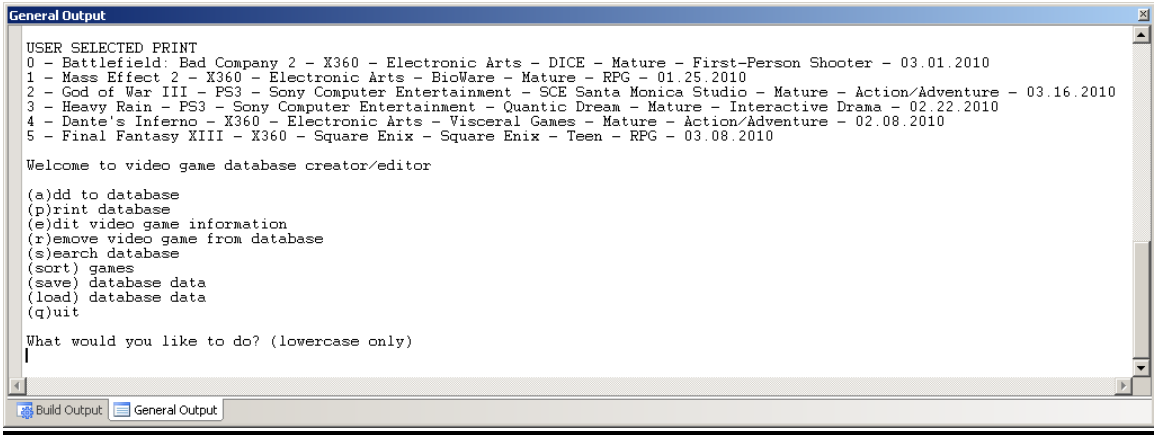
3. Edit video game
    a. From the main menu type “e” and press “Enter”
    b. You will be prompted to input the ID number of the specific video game you would like to edit
    c. Input the ID number and you will be shown the video game you have selected, you will be prompted to confirm if the current video game is the one you would like to edit
    d. From here you will be prompted to choose which attributes you would like to edit, you can choose a specific attribute or select all attributes to edit
    e. Once you make a selection you will be shown the current value of that attribute and prompted to input a new value
    f. Once you have finished editing you will be return to the main menu



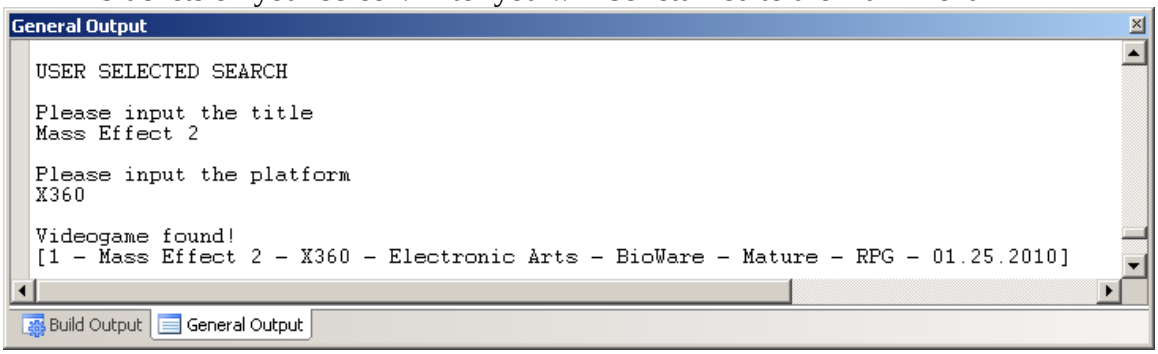
4. Remove video game
    a. From the main menu type “r” and press “Enter”
    b. You will be prompted to the input the ID number of the specific video game and you would like to remove
    c. Input the ID number and you will be shown the video game you have selected, you will be prompted to confirm if the current video game is the on you would like to remove
    d. Once confirmed the video game will be removed from the database and you will be returned to the main menu



5. Print Database
    a. From the main menu type “p” and press “Enter”
    b. If there are entries within the database the database will be printed in sequential order or in the order which it was last sorted

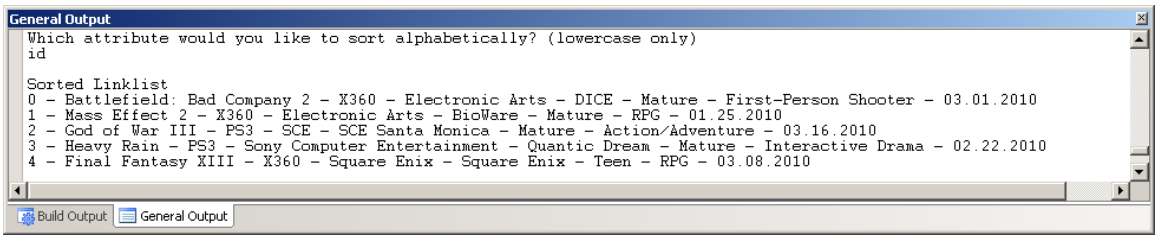


6. Search Database
    a. From the main menu type “s” and press “Enter”
    b. You will be prompted for the title of the video game you are searching for.
    c. After inputting the title you will be prompted for the platform of the video game your searching for
    d. If the video game is found you will be shown all the games information within brackets on your screen. After you will be returned to the main menu

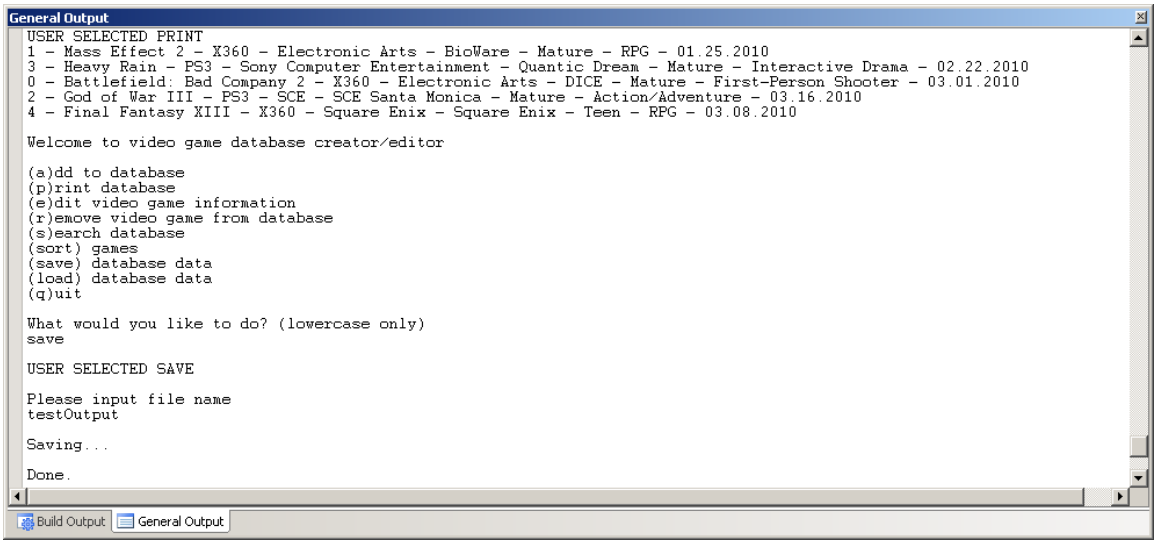


7. Sort Database
    a. From the main menu type “sort” and press “Enter”
    b. You will be prompted with seven options of which attribute you would like to sort by

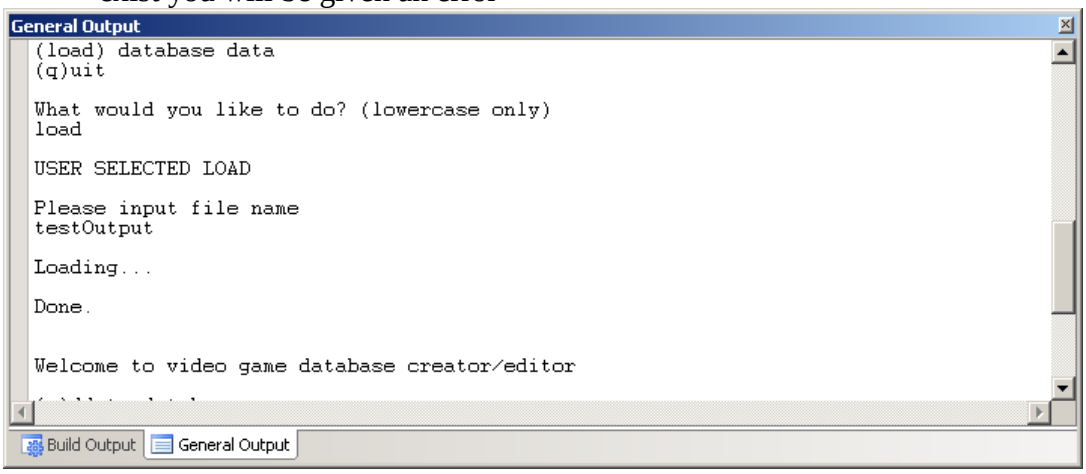
c. Once you have entered the letter(s) corresponding to the option you have chose the database will be sorted, and displayed. After you will be returned to the main menu



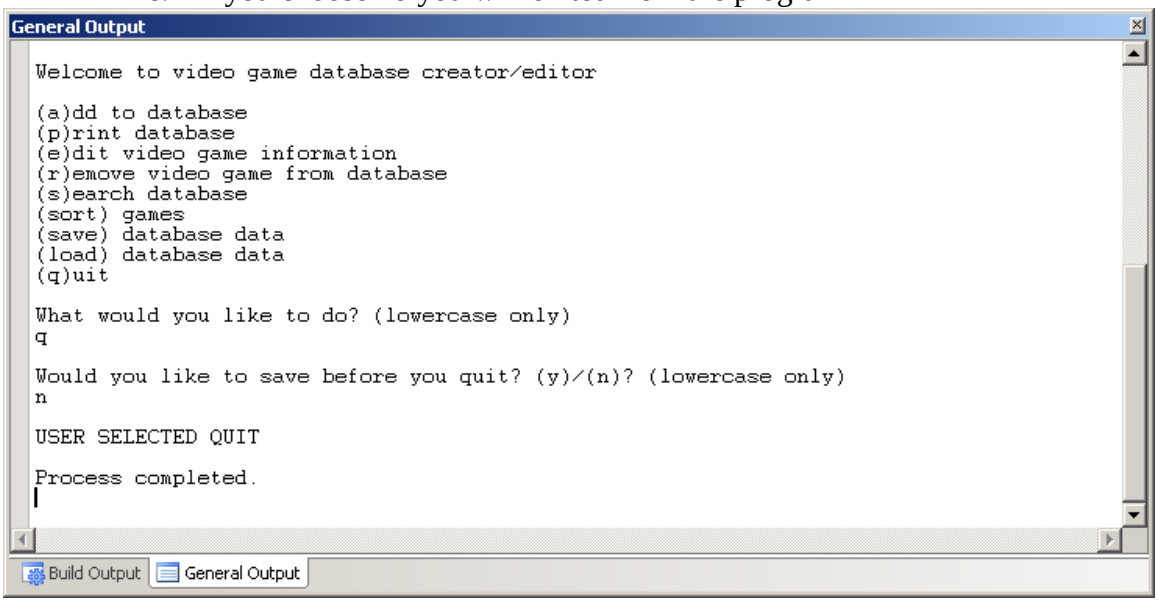
8. Save Database
    a. From the main menu type “save” and press “Enter”
    b. If the database is not empty you will be prompted for a file name
    c. After entering the file name the database will be saved and you will be returned to the main menu



9. Load Database
    a. From the main menu type “load” and press “Enter”
    b. You will be prompted for a file name
    c. If the file exists the database will be loaded into the system, if the file does not exist you will be given an error



10. Quit Editor
    a. From the main menu type “q” and press “Enter”
    b. You will be prompted it you would like to save
    c. If you choose yes you will be prompted for a file name
    d. The file will save and the program will exit
    e. If you choose no you will exited from the program



Mastery Aspects

Mastery Aspect	Where	How	Why
6. Polymorphism	Pgs. 26-28	The VideoGame class has default constructors, which take no parameters, but specific constructors that take multiple parameters	Depending on what attributes are variable to initialize an object, different constructors are used at different times in the program.
8. Encapsulation	Pgs. 26-28	The VideoGame class establishes objects that encapsulate various memory references such as Platform and ESRB Rating	The video games in the database have to contain data to display and manipulate. To contain this data within the object, encapsulation must be used.
9. Parsing a text file or other data stream	Pgs.51; Lines 395-399	In VGDBMain after the user selects to remove a video game from the database, the input string must be parse in order to retrieve the integer needed to initialize the idSearch method	Before removing a video game from the database an identification number must be retrieve from the user in order to ensure the correct video game is removed from the database. When the user inputs the number it can only be accepted as a string, but the remove function will only accept a integer because of the configuration of the attribute. Therefore in order to return the correct type the input string must be parsed.
10. Implementing a hierarchical composite data structure.	Pgs.22	The VGDBMain class contains the getInput class and the VGDB. The VGDB class contains the DB class. The DB class contains the GameNode class. The GameNode class contains the VideoGame class.	In order for the user to be able to use a user-friendly and simple menu interface it was necessary and practical for all six java classes to be linked hierarchically.

Mastery Aspect	Where	How	Why
11. Five SL mastery factors.	**1.** Pgs. 26-28 **2.** Pgs. 44-52 **3.** Pgs. 33-41 **4.** Pgs. 40-41; Lines 443-455 **5.** Pgs. 37-40; Lines 253-440	**1.** User-Defined Objects **2.** Complex selection **3.** Loops **4.** User-Defined Methods w/parameters & return values **5.** Sorting	**1.** User-Defined Objects were used to store organized data of different types **2.** Complex selection was used in the menu system to ensure every options was possible to the user and options that were not possible were stated to the user **3.** Loops were used during several methods that were require to traverse through the linked list until a specific goal was met **4.** idSearch require the specific parameter of an int idNum. It used the idNum to find the ID attribute of a node that equaled idNum, while doing this it kept track of the linked list index it was currently at. When the goal was met it returned the int count (index). **5.** alphaSort allows the user to input a option displayed by a menu within the editor. This option would correspond to a attribute which the user would like the database to be sorted by. After the parameter was put into alphaSort, alphaSort would then return the sorted Database
12-15. Complete implementation of abstract data type (Linked List)	Pgs. 30-41	The DB class is a complete linked list. This complete implementation of the linked list includes the following methods: Insert, remove, print, size, and isEmpty.	A linked list was used in the DB class to organize and modify video game data.