Photo Database

IB Computer Science HL 2 Dossier

Alex Lindeman
Washington-Lee High School

Table of Contents

A1. Analyzing the Problem **4**
- Inputs
- Outputs
- User Interfaces
- User Actions (diagram)

A2. Criteria for Success **6**
- Behavior
- Goals
- System Requirements

A3. Prototype Solution **7**
- Flowchart
- Illustrations

B1. Data Structures **10**
- Data Structure
- Classes
  - Event
  - Picture
  - Tag
  - Database Diagram
- Data Manipulation

B2. Algorithms **12**
- searchTags
- print
- parseMultipleTags

B3. Modular Organization **14**

C1. Programming Style **16**
- DatabaseMain
- Database
- EventList
- EventNode
- Event
- PictureList
- PictureNode
- Picture
- TagList
- TagNode
- Tag

C2. Usability **44**

C3. Handling Errors **45**

C4. Success of Program **46**

D1. Test Output **47**

D2. Solution Evaluation **51**
- Functionality
- Efficiency
- Limitations
- Design

D3. User Manual **52**
- Installation

Using the Menu System
Importing Pictures
Single Picture
Folder of Pictures
Searching for Pictures by Tag
Browsing Events
Mastery Aspects 57

A1. Analyzing the Problem

Many people – photographers, travellers, families – have large collections of pictures. Unfortunately, with these larger collections of pictures it can be difficult to find the one to show off to family, print out, edit, etc.. This can make it annoying and cumbersome for many people who store pictures on their computer, especially for people who aren't as familiar with computers and how to store data efficiently.

The problem at hand is that these people need a way to organize their photos that is easy to use and not too complicated. To solve this, a program that I would create would be able to have collections of pictures with tags, names, and other metadata that would allow them to be sorted, searched, printed, and edited easily.

This problem has already been addressed by programs such as iPhoto, or on the Internet with Flickr. These programs allow users to add tags and names to their images and make searching easier. This solution intends to be much “lighter”, ie. less cumbersome and complicated to use.

Problems with ways many users store photos now is that they, well, don't. They have them cluttered up in the My Pictures folder, or worse, have them scattered around their hard drive on their desktop and in random folders. This program intends to store them in a way that is easy to use whether you use the program once or a thousand times.

Inputs

- Importing pictures
  - a single picture
    - some tags which describe the picture, like “dog” or “umbrella”
    - a collection (“event”) to “put” the pictures in, like “Beach Trip 2008”
  - a folder of pictures
    - the name of the event
- Searching for pictures
  - by tags, i.e. search for “car” will bring back evey picture that has the “car” tag
  - by event, which will show every photo inside of the event

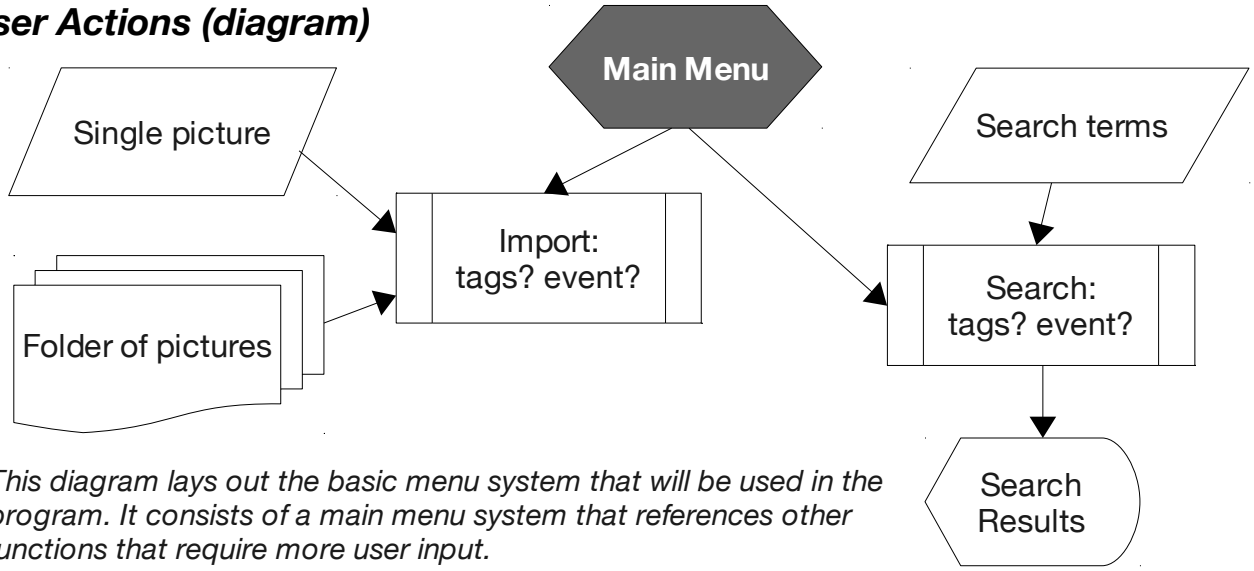
Outputs

- Search results
  - A list of matching pictures
    - why they've been matched (bold text, like “Tags: umbrella, **dog**, chair, beach”)
    - manipulation options, like editing and deleting the picture

User Interfaces

- Command line-based menu and browser system

User Actions (diagram)



This diagram lays out the basic menu system that will be used in the program. It consists of a main menu system that references other functions that require more user input.

A2. Criteria for Success

This section is meant to outline a photo organization system that will be easy to use for inexperienced computer users with vital functions they frequently use.

Behavior

The system will:

- have an intuitive, easy-to-use menu system
- allow the user to add, edit, and remove pictures
- have a search process to find pictures quickly
- be able to retrieve data in the filesystem and copy it to an easy-to-find location, like the Desktop

My program aims to solve the problem which many novice computer users have of storing their digital photos. They tend to randomly store them throughout their computer, on their desktop, or with clunky and bloated software that came with their camera. This program intends to solve these problems by being easy to use and able to organize photos automatically.

Goals

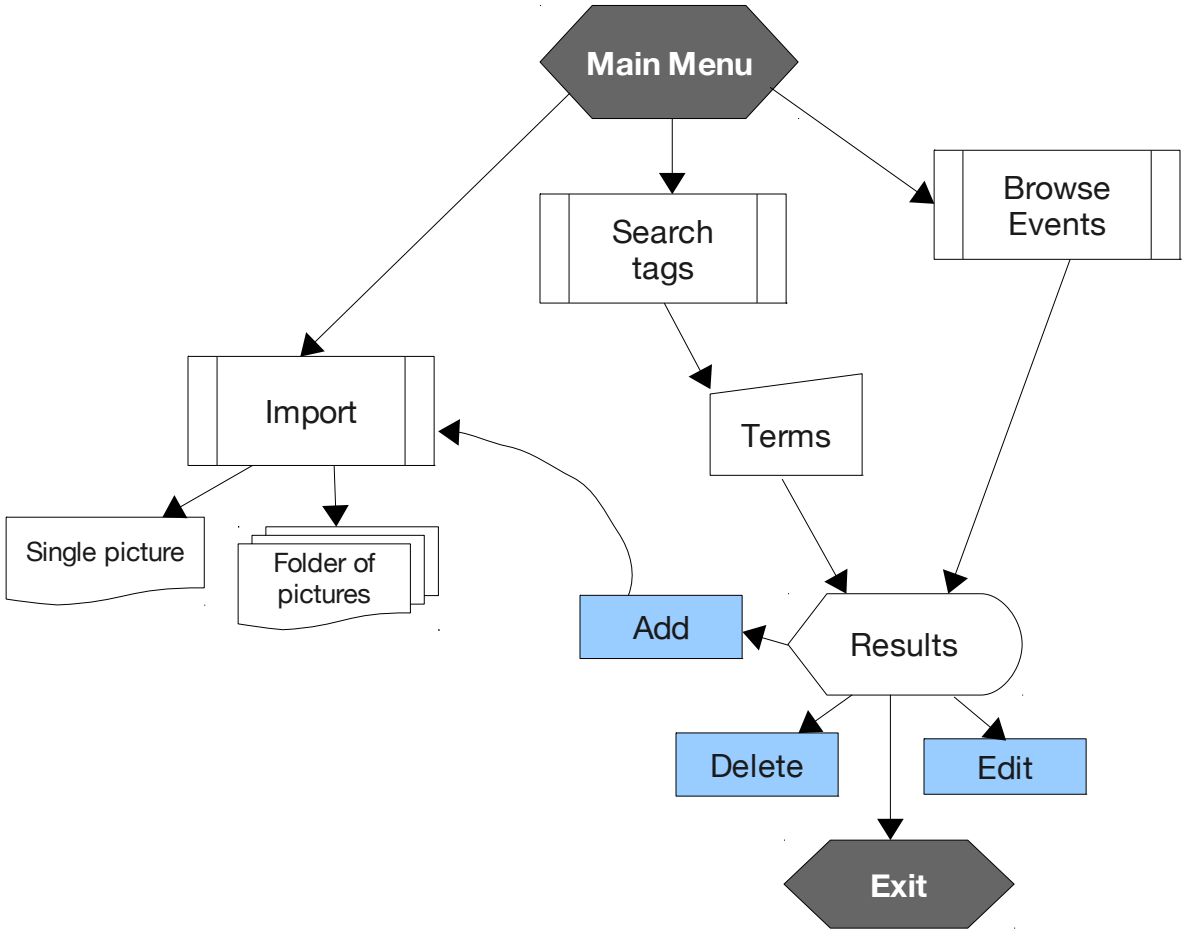
1. Create a menu system that is easy to use for even the most inexperienced computer users to use.
2. Allow the user to import photos with minimal input.
3. Allow data manipulation through search results, also with minimal input.
4. Use a fast search method to ease or prevent frustration waiting for searches to complete.
5. Allow the user to transfer a picture to the Desktop to easily email and print it.
6. Exclude any unnecessary features that would confuse the user or slow the program down.

System Requirements

- A computer running Ubuntu Linux with the Java Runtime Environment
- An xterm or VT100-compatible terminal
- Enough hard drive space and memory to store pictures (the more pictures the user has, the more space is needed)
- A reasonably fast CPU to search through pictures quickly

A3. Prototype Solution

Flowchart



Illustrations

```
Main Menu

Type a number corresponding to the menu
entry and press Enter to confirm

[1] Import single picture
[2] Import folder

[3] Search by tag
[4] Browse events


[0 Exit]
>> _
```

```
Exit

Are you sure you want to quit?

[1 No]
[0 Yes]

>> _
```

```
Import folder of pictures

Enter a name for the event:
Washington, DC 2010

Enter the folder you wish to import (you
can drag and drop from the file explorer
in Ubuntu):
'/home/alex/dossier/test/dc/'

Pictures found: 6
Importing. This may take a moment [done]
[0 Return to main menu]
>> _
```

```
Import single picture

Enter the path to the picture (you can
drag and drop from the file explorer in
Ubuntu):
'/home/alex/Desktop/IMG_3008.jpg'

Enter the event for this picture:
Washington, DC 2010

Enter tags that describe the picture:
monument, lincoln, statue, history

Importing. [done]
[0 Return to main menu]
>> _
```

```
Search by tag

Enter one tag to search for and press
Enter to begin searching:
>> _
```

```
Search results for tag “vacation”:

[1] pic001.jpg in “Bahamas 2002”
[2] pic002.jpg in “Toronto 2005”
[3] pic003.jpg in “South Carolina 1999”
[4] pic004.jpg in “Paris 2007”
[5] pic005.jpg in “Florida 2009”

[0 Exit to main menu]
>> _
```

```
Browsing events:

[1] Toronto 2005
[2] South Carolina 1999
[3] Paris 2007
[4] Florida 2009

[0 Exit to main menu]
>> _
```

```
Browsing event “Toronto 2005”:

[1] pic001.jpg
[2] pic002.jpg
[3] pic003.jpg
[4] pic004.jpg
[5] pic005.jpg

[6 Back to Events] [0 Exit to main menu]
>> _
```

```
Picture details for pic001.jpg

Event: Bahamas 2002
Tags: vacation, beach, fun, family

Location:
/home/alex/dossier/test/pic001.jpg

[1] Copy to Desktop

[2] Edit details
[2] Delete

[9 Back to search results]
[0 Exit to main menu]
>> _
```

```
Edit picture tags

Current values: “vacation, beach, fun,
family”

Enter a new value and press Enter to
confirm
>> _
```

Delete “pic001.jpg” Are you sure you want to delete this picture? This action CANNOT be undone. [yes Delete] >> _	Delete “Bahamas 2002” Deleting this event will ALSO delete every picture associated with that event. Are you sure you want to delete this event? This action CANNOT be undone. [yes Delete] >> _

B1. Data Structures

Data Structure

The program will use three Linked Lists to store its data. Linked Lists are optimal because the database needs to be dynamically sized (eliminating Arrays). Although Binary Trees would have been another alternative to the problem, they use numbers to sort themselves and they way this program's database is arranged, a Binary Tree has no advantage of speed over a Linked List.

Classes

Event

Properties	**Type**
Name	String
Pictures	LinkedList

Example	Beach 2006	[1, 2, 3, 4, 5]

Picture

Properties	**Type**
URI	String
Tags	LinkedList

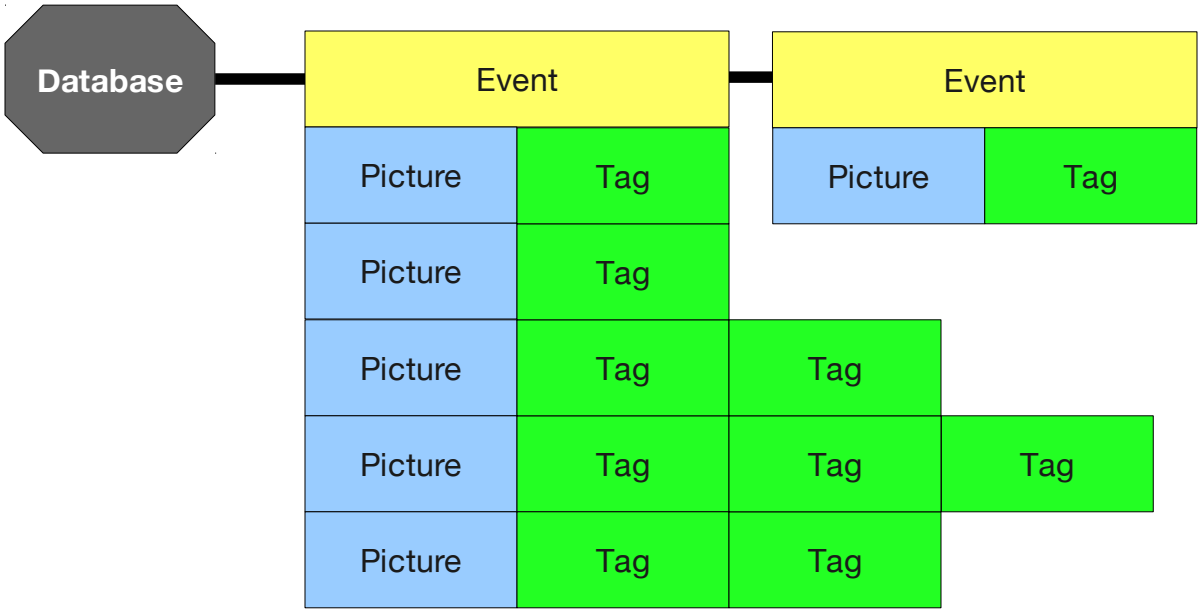
Example	/home/alex/picture.jpg	[umbrella, dog, sand, ocean]

Tag

Properties	**Type**
Name	String

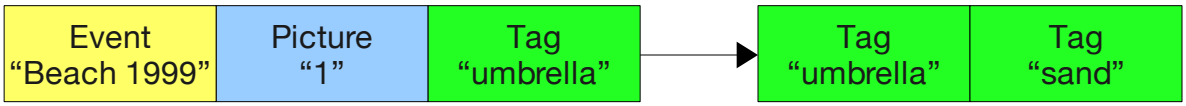
Example	umbrella

Database Diagram

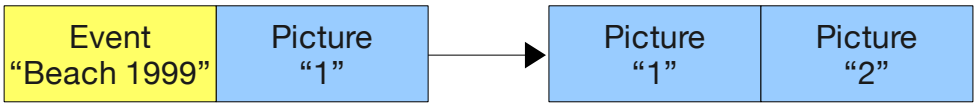


Data Manipulation

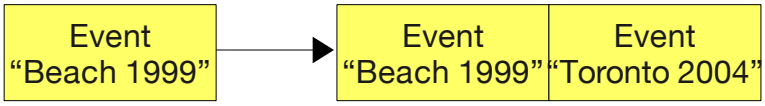
Adding tag:



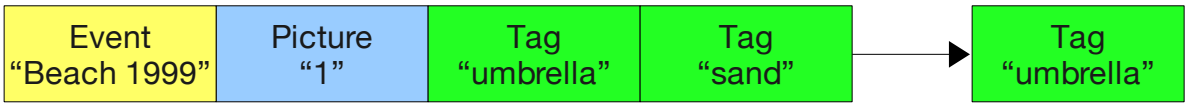
Adding picture:



Adding event:

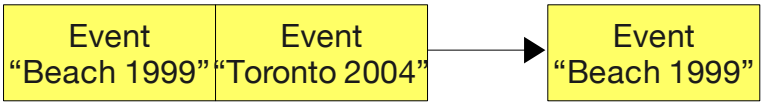


Removing tag:



Removing picture:

Removing event:



B2. Algorithms

searchTags

Description	Searches through the list of tags associated with a picture.
Parameters	Name of a tag *(String)*
Returns	LinkedList of pictures that are associated with the tag.
Pre-conditions	At least 1 tag exists, and there is at least 1 picture in the database
Post-conditions	Something was found
Code	1. Checks if the tag exists 2. Searches through the LinkedList of pictures that are associated with the tag 3. Adds matches to a new LinkedList 4. Returns the new LinkedList

print

Description	Prints every element of the LinkedList into the terminal window.
Parameters	*None*
Returns	Nothing
Pre-conditions	The LinkedList is not empty
Post-conditions	None
Code	1. Check if the LinkedList is empty or not 2. Go through each element in the LinkedList and print it to the terminal window

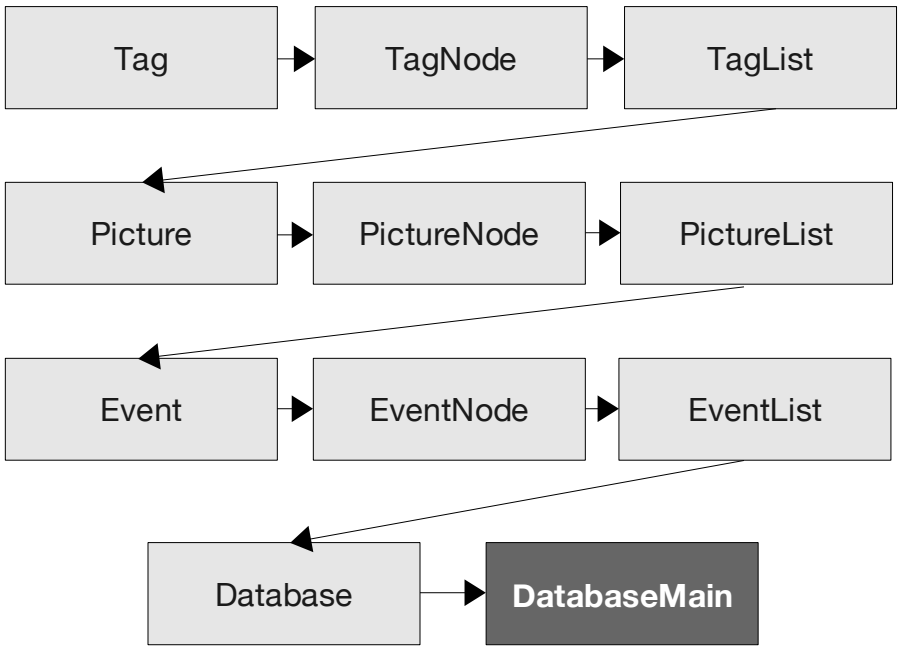
parseMultipleTags

Description	Searches through the list of tags associated with a picture.
Parameters	List of tags separated by commas *(String)*
Returns	A TagList of the separated tags
Pre-conditions	The TagList object exists
Post-conditions	None

Code	1. Get the string of tags which are separated by commas 2. Split the string into an array 3. Go through each element of the array and add it to a new TagList 4. Return the completed TagList

addTag (tagName)	Adds a tag to a picture
removeTag (tagName)	Removes a tag from a picture
hasTag (tagName)	Returns boolean if picture has the specified tag
addEvent (eventName)	Creates a new event
removeEvent (eventName)	Deletes an event

B3. Modular Organization



Tag	Essentially a string which acts as a “keyword” for the contents of a picture, describing its contents, style, or anything descriptive of the image that the user chooses.
TagNode	A container for the **Tag** class which allows it to be used in a LinkedList.
TagList	A LinkedList which contains multiple instances of the **TagNode** and **Tag** class so the user can describe the picture in more than one keyword.
Picture	Contains a URI string so that the picture can be located in the system, and contains a **TagList** which relate to the picture's appearance to allow the user to search for them easily.
PictureNode	A container for the **Picture** class which allows it to be used in a LinkedList.
PictureList	A LinkedList which contains many **Pictures** so the user can sort them into categories, or **Events**.
Event	Contains a name so the user can generalize where all the pictures are from (ie. a category) and a **PictureList** which holds all of the pictures related to that category.
EventNode	A container for the **Event** class which allows it to be used in a LinkedList.

EventList	A LinkedList made up of **Events** so that many may be organized in one single element.
Database	A container for the **EventList** class, which provides additional functionality and important methods for the main file.
DatabaseMain	The file which is directly run. It contains the menu system and functions which manipulate the database from the end user's actions.

C1. Programming Style

DatabaseMain

```
/*
* DatabaseMain.java
* It's the file you run!
*
* Author: Alex Lindeman
* School: Washington-Lee High School
* Modified: Mar 19 2010
*
* IDE: gedit 2.28.0
* System: IBM ThinkPad X31
*/

import java.io.*;

class DatabaseMain {

   private static BufferedReader stdin =
     new BufferedReader(new InputStreamReader(System.in));

   private static int getNumericalInput(String prompt) throws IOException {
     while (true){
       try {
         System.out.print(prompt);
         String userInput = stdin.readLine();
         return Integer.parseInt(userInput);
       } catch (NumberFormatException error) {
         System.out.println("Invalid selection. Enter a number that corresponds to
a menu entry and try again.");
       }
     }
   }

   private static TagList parseMultipleTags(String tags){
     String [] tagsListArray = null;
     tagsListArray = tags.split(",");
     TagList tagLL = new TagList();
     for (String tag : tagsListArray){
       Tag tagToAdd = new Tag(tag);
       tagLL.add(tagToAdd);
     }
     return tagLL;
   }

   public static void main(String[] args) throws IOException {

     boolean exit = false;
     int choice;
     Database db = new Database();
```

```
    while (!exit){
      System.out.println("Main Menu" + "\n" +
      "Type a number corresponding to the menu entry and press Enter to confirm"
      + "\n\n" +
      "[1] Import single picture" + "\n" +
      "[2] Import folder" + "\n\n" +
      "[3] Search by tag" + "\n" +
      "[4] Browse events" + "\n\n" +
      "[0 Exit]" + "\n");

      choice = getNumericalInput(">> ");

      switch(choice){
        case 1:
          // ask for user input
          System.out.println("Import single picture" + "\n\n" +
          "Enter the path to the photo, or drag and drop from the file
explorer:");
          String uri = stdin.readLine();
          System.out.println("\n" + "Enter the event you wish to add the picture
to:");
          String event = stdin.readLine();
          System.out.println("\n" + "Enter some tags which describe the image,
separated by commas:");
          String tags = stdin.readLine();
          System.out.print("\n" + "Importing...");

          // Strip quotes and trailing space from drag + drop
          int sl = uri.length();
          if (
          (uri.indexOf("'") == 0 && uri.substring(sl-1,sl).indexOf("'") == 0) ||
          (uri.indexOf("'") == 0 && uri.substring(sl-2,sl-1).indexOf("'") == 0)
          ){
            uri = uri.substring(1, sl-2);
          }

          // import
          TagList tagLL = parseMultipleTags(tags);
          Event eventToAdd = new Event(event);
          Picture picToAdd = new Picture(uri, tagLL);
          db.addPicture(eventToAdd, picToAdd);

          // end
          System.out.println("\n" + "[Press Enter to return to the main menu]");
          stdin.readLine();
          break;

        case 2:
          // User input
          System.out.println("Import folder as event" + "\n\n" +
          "Enter the full path of the folder you wish to import:");
          String path = stdin.readLine();
          System.out.println("\n" + "Enter a name for the event:");
          String newEventName = stdin.readLine();
```

```
            // Create the event
            Event newEvent = new Event(newEventName);

            // Strip any begin/end quotes from pathname in addition to any
trailing space to it
            int pl = path.length();
            if (
            (path.indexOf("'") == 0 && path.substring(pl-1,pl).indexOf("'") == 0)
||
            (path.indexOf("'") == 0 && path.substring(pl-2,pl-1).indexOf("'") ==
0)
            ){
              path = path.substring(1, pl-2);
            }

            // Parse the folder for jpegs, tiffs, and pngs
            File dir = new File(path);
            String[] contents = dir.list();
            if (contents == null) {
              System.out.println("\n" + "The specified folder (\"" + path +
              "\") does not exist or is not a valid directory.");
            } else {
              System.out.print("\n" + "Pictures found: " + contents.length + "\n"
+ "Importing. This may take a moment ");
              for (int i = 0; i < contents.length; i++) {
                // Ignore things that aren't pictures
                if (contents[i].indexOf(".jpg") != -1 ||
contents[i].indexOf(".JPG") != -1){
                  // Add them to the temporary event
                  Picture newPicture = new Picture (path + contents[i], null);
                  newEvent.addPicture(newPicture);
                }
              }
              // Add them as a new event
              db.addEvent(newEvent);
              System.out.println("[done]");
            }

            // end
            System.out.println("\n" + "[Press Enter to return to the main menu]");
            stdin.readLine();
            break;

          case 3:
            // User input
            System.out.println("Search by tag" + "\n\n" +
            "Enter a tag to search for:");
            System.out.print(">> ");
            String term = stdin.readLine();
            System.out.println("\n" + "Search results:");
            Tag searchterm = new Tag(term);
            Picture tempPic;
            PictureList results = new PictureList();
            for (int i = 0; i < db.events.getSize(); i ++){
              PictureList searchThis = db.events.getEvent(i).getPictureList();
```

```
            for (int j = 0; j < searchThis.getSize(); j ++){
              tempPic = searchThis.getPicture(j);
              if (tempPic.getTag(term) != null){
                results.add(tempPic);
              }
            }
          }
          results.print();
          int pictureChoice = getNumericalInput("\n" + "[0 Main menu]" +
          "\n" + ">> ") -1;
          if (pictureChoice >= 0){
            boolean exitEditPic = false;
            while (!exitEditPic){
              // editing a picture
              Picture toEdit = results.getPicture(pictureChoice);
              System.out.println("Picture details for " + toEdit + ":" + "\n\n"
+
              "Event: " + results);
              if (toEdit.getTagsAsLL() != null){
                System.out.println("Tags: " + toEdit.getAllTags() + "\n");
              } else {
                System.out.println("No tags" + "\n");
              }
              System.out.println("Location:" + "\n" + toEdit + "\n\n" +
              "[1] Copy to Desktop" + "\n\n" +
              "[2] Edit tags" + "\n" +
              "[3] Delete" + "\n\n" +
              "[0 Main menu]");

              // ask for choice
              int detailChoice = getNumericalInput(">> ");
              switch (detailChoice){
                case 1:
                  try {
                    File pic = new File(toEdit.uri);
                    File desk = new File("~/Desktop/"+toEdit.getBase());
                    FileReader in = new FileReader(pic);
                    FileWriter out = new FileWriter(desk);
                    int c;
                    while ((c = in.read()) != -1)
                      out.write(c);

                    in.close();
                    out.close();
                  } catch (IOException x) {
                    System.out.println("The picture could not copied to the
desktop.");
                  }
                  break;
                case 2:
                  System.out.println("\n" + "Edit Tags" + "\n\n" +
                  "Enter a new list of tags to associate with the image,
separated by commas:");
                  String newtags = stdin.readLine();
                  TagList newTags = parseMultipleTags(newtags);
```

```
                    toEdit.changeTags(newTags);
                    System.out.println("Tags changed.");
                    break;
                  case 3:
                    System.out.println("\n" + "This will delete this picture
forever." + "\n" +
                    "Are you sure you want to delete this picture?" + "\n\n" +
                    "[4 Delete]" + "\n" + "[0 No]");
                    int verify = getNumericalInput(">> ");
                    if (verify == 4){
                      db.events.remove(pictureChoice);
                      System.out.println("Picture deleted.");
                    }
                    break;
                  case 0:
                  default:
                    exitEditPic = true;
                    break;
                  }
                }
              }

            // end
            System.out.println("\n" + "[Press Enter to return to the main menu]");
            stdin.readLine();
            break;

          case 4:
            boolean stop = false;
            System.out.println("Browse events");
            System.out.println("Enter an event to browse:" + "\n");
            db.listEvents();
            int eventChoice = getNumericalInput("\n" + "[0 Main menu]" +
            "\n" + ">> ") - 1;

            if (eventChoice >= 0){
              Event result = db.getEvent(eventChoice);
              result.listAllPictures();
              pictureChoice = getNumericalInput("\n" + "[0 Main menu]" +
              "\n" + ">> ") -1;

              if (pictureChoice >= 0){
                boolean exitEditPic = false;
                while (!exitEditPic){
                  // editing a picture
                  Picture toEdit = result.getPicture(pictureChoice);
                  System.out.println("Picture details for " + toEdit + ":" +
"\n\n" +
                  "Event: " + result);
                  if (toEdit.getTagsAsLL() != null){
                    System.out.println("Tags: " + toEdit.getAllTags() + "\n");
                  } else {
                    System.out.println("No tags" + "\n");
                  }
                  System.out.println("Location:" + "\n" + toEdit + "\n\n" +
```

```
                    "[1] Copy to Desktop" + "\n\n" +
                    "[2] Edit tags" + "\n" +
                    "[3] Delete" + "\n\n" +
                    "[0 Main menu]");

                    // ask for choice
                    int detailChoice = getNumericalInput(">> ");
                    switch (detailChoice){
                       case 1:
                         try {
                               File pic = new File(toEdit.uri);
                               File desk = new File("~/Desktop/"+toEdit.getBase());
                               FileReader in = new FileReader(pic);
                               FileWriter out = new FileWriter(desk);
                               int c;
                               while ((c = in.read()) != -1)
                                 out.write(c);

                               in.close();
                               out.close();
                         } catch (IOException x) {
                               System.out.println("The picture could not copied to
the desktop.");
                         }
                         break;
                       case 2:
                         System.out.println("\n" + "Edit Tags" + "\n\n" +
                         "Enter a new list of tags to associate with the image,
separated by commas:");
                         String newtags = stdin.readLine();
                         TagList newTags = parseMultipleTags(newtags);
                         toEdit.changeTags(newTags);
                         System.out.println("Tags changed.");
                         break;
                       case 3:
                         System.out.println("\n" + "This will delete this picture
forever." + "\n" +
                         "Are you sure you want to delete this picture?" + "\n\n" +
                         "[4 Delete]" + "\n" + "[0 No]");
                         int verify = getNumericalInput(">> ");
                         if (verify == 4){
                               db.events.remove(pictureChoice);
                               System.out.println("Picture deleted.");
                         }
                         break;
                       case 0:
                       default:
                         exitEditPic = true;
                         break;
                       }
                    }
                 }
              }
            // exit
            System.out.println("\n" +
```

```
            "[Press Enter to return to the main menu]");
            stdin.readLine();
            break;
          case 0:
          default:
            System.out.println("Are you sure you want to quit?"
            + "\n\n" +
            "[0 Quit]" + "\n" +
            "[1 Main menu]");
            choice = getNumericalInput(">> ");
            if (choice != 1){
              exit = true;
            }
            break;
        }
      }
    }
}
```

Database

```
/*
* Database.java
* Consolidates Tags, Pictures, and Events into a single database.
*
* Author:   Alex Lindeman
* School:   Washington-Lee High School
* Modified: Mar 17 2010
*
* IDE:      gedit 2.28.0
* System:   IBM ThinkPad X31
*/

class Database {

      // globals
      public EventList events = new EventList();

      // constuctor
      Database (){ }

      Database (EventList newEventsList){
            events = newEventsList;
      }

      // pictures
      public void addPicture(Event event, Picture pic){
            // Convert to string first because it's easy
            String eventStr = event.toString();
            int result = events.find(eventStr);

            // See if event exists first
            if (events.find(eventStr) >= 0){
                  Event retrieved = events.getEvent(result);
                  retrieved.addPicture(pic);
                  events.add(retrieved);
            }
            // If it doesn't, then make it
            else {
                  Event newEvent = event;
                  newEvent.addPicture(pic);
                  events.add(newEvent);
            }
      }
      public void removePicture(Event event, int index){
            String eventStr = event.toString();
            int result = events.find(eventStr);
            if (events.getEvent(result) != null){
                  events.remove(index);
            }
      }

      // events
```

```
	public void addEvent(Event newEvent){
		events.add(newEvent);
	}
	public Event removeEvent(int index){
		return events.remove(index);
	}
	public Event getEvent(int index){
		return events.getEvent(index);
	}
	public void listEvents(){
		events.print();
	}
}
```

EventList

```
/*
* EventList.java
* LinkedList container for Events.
*
* Author:   Alex Lindeman
* School:   Washington-Lee High School
* Modified: Mar 3 2010
*
* IDE:      gedit 2.28.0
* System:   IBM ThinkPad X31
*/

class EventList {

      // globals
      private EventNode head = null;
      private EventNode tail = null;
      private int size = 0;

      // constructor
      EventList (){ }

      // checks if list is empty
      public boolean isEmpty (){
            return size == 0;
      }

      // returns the size
      public int getSize (){
            return size;
      }

      // gets a single eventture at a specific index
      public Event getEvent (int index){
            int count = 0;
            for(EventNode current = head; current != null; current = current.next){
                  if (count == index){
                        return current.event;
                  }
                  count ++;
            }
            return null;
      }

      // gets all events
      public void print(){
            int pos = 1;
            for(EventNode currentNode = head; currentNode != null;
            currentNode = currentNode.next){
                  System.out.println("[" + pos + "] "+ currentNode.event);
                  pos ++;
            }
```

```
}

// search method
public int find(String searchterm){
      int count = 0;
      for (EventNode current = head; current != null;
      current = current.next){
            if (current.event.name.equals(searchterm))
                  return count;
            count ++;
      }
      return -1;
}

// appends a eventture to the end of the list
public void add (Event newEvent){
      EventNode temp = new EventNode();
      temp.event = newEvent;

      if (isEmpty()){
            head = temp;
            tail = temp;
      } else {
            tail.next = temp;
            tail = temp;
      }

      size ++;
}

// adds a event to a specific index
public void add (Event newEvent, int index){

      // adding to empty
      if (isEmpty()) {
            EventNode temp = new EventNode();
            temp.event = newEvent;
            head = temp;
            tail = temp;
            size ++;
      }

      // adding to beginning
      if (!isEmpty() && index == 0 )
      {
            EventNode temp = new EventNode();
            temp.event = newEvent;
            temp.next = head;
            head = temp;
            size ++;
      }

      // adding to end
      if (!isEmpty() && index == size)
      {
```

```
            EventNode temp = new EventNode();
            temp.event = newEvent;
            tail.next = temp;
            tail = temp;
            size ++;
      }

      // adding to middle
      if (!isEmpty() && index < size && index > 0){
            int count = 0;
            EventNode prev = head;
            for(EventNode current = head; current != null;
            current = current.next){
                  if (count == index){
                        EventNode temp = new EventNode();
                        temp.event = newEvent;
                        temp.next = current;
                        prev.next = temp;
                  }
                  prev = current;
                  count ++;
            }
            size ++;
      }

      // adding to non-existent index
      if (!isEmpty() && index > size){
            System.out.println("Could not add event " + newEvent +
            " to index " + index + " (out of bounds)");
      }
}

// removes a eventture at a particular index
public Event remove (int index){

      // invalid index
      if (index >= size || index < 0) return null;

      // removing from empty list
      if (isEmpty()) return null;

      // removing last remaining event
      if (index == 0 && index != size - 1){
            Event temp = head.event;
            head = null;
            tail = null;
            size --;
            return temp;
      }

      // removing first event (head)
      if (index == 0 && head != tail){
            Event temp = head.event;
            head = head.next;
            size --;
```

```
                return temp;
            }

            // removing last event (tail)
            if (index == size - 1 && size > 1){
                Event temp = tail.event;
                for (EventNode current = head; current != null;
                current = current.next){
                    if (current.next == tail){
                        tail = current;
                        tail.next = null;
                        size --;
                        return temp;
                    }
                }
                return temp;
            }

            // removing from middle
            if (index != 0 && index != size -1){
                int count = 0;
                EventNode prev = head;
                for (EventNode current = head; current != null;
                current = current.next){
                    if (count == index){
                        Event temp = current.event;
                        prev.next = current.next;
                        size --;
                        return temp;
                    }
                    prev = current;
                    count ++;
                }
                return null;
            }

            // for some reason nothing matches, do nothing
            return null;
        }
}
```

EventNode

```
/*
* EventNode.java
* Class definition for Event nodes (used in LinkedList).
*
* Author:   Alex Lindeman
* School:   Washington-Lee High School
* Modified: Mar 3 2010
*
* IDE:      gedit 2.28.0
* System:   IBM ThinkPad X31
*/

class EventNode {

      // globals
      public Event event = null;
      public EventNode next = null;

      // constructor
      EventNode (){ }

}
```

Event

```
/*
* Event.java
* Class definition for Events.
*
* Author:   Alex Lindeman
* School:   Washington-Lee High School
* Modified: Mar 3 2010
*
* IDE:      gedit 2.28.0
* System:   IBM ThinkPad X31
*/
class Event {

      // globals
      public String name;
      public PictureList pix;

      // construct
      Event (){
      }

      Event (String newName){
            name = newName;
            pix = new PictureList();
      }

      // as string
      public String toString(){
            return name;
      }

      public String getName(){
            return name;
      }

      // picture functions
      public void addPicture(Picture newPicture){
            pix.add(newPicture);
      }
      public Picture removePicture(int index){
            return pix.remove(index);
      }
      public Picture getPicture(int index){
            return pix.getPicture(index);
      }
      public PictureList getPictureList(){
            return pix;
      }
      public void listAllPictures(){
            pix.print();
      }
}
```

PictureList

```
/*
* PictureList.java
* LinkedList container for Pictures.
*
* Author:   Alex Lindeman
* School:   Washington-Lee High School
* Modified: Mar 3 2010
*
* IDE:      gedit 2.28.0
* System:   IBM ThinkPad X31
*/

class PictureList {

      // globals
      private PictureNode head = null;
      private PictureNode tail = null;
      private int size = 0;

      // constructor
      PictureList (){ }

      // checks if list is empty
      public boolean isEmpty (){
            return size == 0;
      }

      // returns the size
      public int getSize (){
            return size;
      }

      // gets a single picture at a specific index
      public Picture getPicture(int index){
            int count = 0;
            for(PictureNode current = head; current != null;
            current = current.next){
                  if (count == index){
                        return current.pic;
                  }
                  count ++;
            }
            return null;
      }

      // gets all pics
      public void print(){
            int pos = 1;
            for (PictureNode currentNode = head; currentNode != null;
            currentNode = currentNode.next){
                  System.out.println("[" + pos + "] "+ currentNode.pic);
                  pos ++;
```

```
	}
}

// search method
public int find(String searchterm){
	int count = 0;
	for (PictureNode current = head; current != null;
	current = current.next){
		if (current.pic.uri.equals(searchterm))
			return count;
		count ++;
	}
	return -1;
}

// appends a picture to the end of the list
public void add (Picture newPicture){
	PictureNode temp = new PictureNode();
	temp.pic = newPicture;

	if (isEmpty()){
		head = temp;
		tail = temp;
	} else {
		tail.next = temp;
		tail = temp;
	}

	size ++;
}

// adds a picture to a specific index
public void add (Picture newPicture, int index){

	// adding to empty
	if (isEmpty()) {
		PictureNode temp = new PictureNode();
		temp.pic = newPicture;
		head = temp;
		tail = temp;
		size ++;
	}

	// adding to beginning
	if (!isEmpty() && index == 0 )
	{
		PictureNode temp = new PictureNode();
		temp.pic = newPicture;
		temp.next = head;
		head = temp;
		size ++;
	}

	// adding to end
	if (!isEmpty() && index == size)
```

```
	{
		PictureNode temp = new PictureNode();
		temp.pic = newPicture;
		tail.next = temp;
		tail = temp;
		size ++;
	}

	// adding to middle
	if (!isEmpty() && index < size && index > 0){
		int count = 0;
		PictureNode prev = head;
		for(PictureNode current = head; current != null;
		current = current.next){
			if (count == index){
				PictureNode temp = new PictureNode();
				temp.pic = newPicture;
				temp.next = current;
				prev.next = temp;
			}
			prev = current;
			count ++;
		}
		size ++;
	}

	// adding to non-existent index
	if (!isEmpty() && index > size){
		System.out.println("Could not add pic " + newPicture +
		" to index " + index + " (out of bounds)");
	}
}

// removes a picture at a particular index
public Picture remove (int index){

	// invalid index
	if (index >= size || index < 0) return null;

	// removing from empty list
	if (isEmpty()) return null;

	// removing last remaining pic
	if (index == 0 && index != size - 1){
		Picture temp = head.pic;
		head = null;
		tail = null;
		size --;
		return temp;
	}

	// removing first pic (head)
	if (index == 0 && head != tail){
		Picture temp = head.pic;
		head = head.next;
```

```
                size --;
                return temp;
            }

            // removing last pic (tail)
            if (index == size - 1 && size > 1){
                Picture temp = tail.pic;
                for (PictureNode current = head; current != null;
                current = current.next){
                    if (current.next == tail){
                        tail = current;
                        tail.next = null;
                        size --;
                        return temp;
                    }
                }
                return temp;
            }

            // removing from middle
            if (index != 0 && index != size -1){
                int count = 0;
                PictureNode prev = head;
                for (PictureNode current = head; current != null;
                current = current.next){
                    if (count == index){
                        Picture temp = current.pic;
                        prev.next = current.next;
                        size --;
                        return temp;
                    }
                    prev = current;
                    count ++;
                }
                return null;
            }

            // for some reason if nothing matches, do nothing
            return null;
        }
}
```

PictureNode

```
/*
* PictureNode.java
* Class definition for Picture nodes (used in LinkedList).
*
* Author:   Alex Lindeman
* School:   Washington-Lee High School
* Modified: Mar 3 2010
*
* IDE:      gedit 2.28.0
* System:   IBM ThinkPad X31
*/

class PictureNode {

      // globals
      public Picture pic = null;
      public PictureNode next = null;

      // constructor
      PictureNode (){ }

}
```

Picture

```
/*
* Picture.java
* Class definition for Pictures.
*
* Author:   Alex Lindeman
* School:   Washington-Lee High School
* Modified: Mar 3 2010
*
* IDE:      gedit 2.28.0
* System:   IBM ThinkPad X31
*/

class Picture {

      // globals
      public String uri;
      private TagList tags;

      // construct
      Picture (){
      }

      Picture (String newURI, TagList newTags){
            uri = newURI;
            tags = newTags;
      }

      // as string
      public String toString(){
            return uri;
      }

      // get filename only
      public String getBase(){
            int slash = uri.lastIndexOf("/");
            int length = uri.length();
            String filename = uri.substring(slash,length);
            return filename;
      }

      // tag functions
      public void addTag(Tag newTag){
            tags.add(newTag);
      }
      public boolean removeTag(String rmTag){
            // remove() won't remove by name, so search for the tag first
            int position = tags.find(rmTag);
            if (position >= 0){
                  tags.remove(position);
                  return true;
            } else {
                  return false;
```

```
            }
      }
      public Tag getTag(String get){
            int pos = tags.find(get);
            if (pos >= 0){
                  return tags.getTag(pos);
            }
            return null;
      }
      public TagList getTagsAsLL(){
            return tags;
      }
      public String getAllTags(){
            tags.print();
            return null;
      }
      public void changeTags(TagList newTags){
            int total = tags.getSize();
            for (int i = 0; i < total; i ++){
                  tags.remove(i);
            }
            tags = newTags;
      }
}
```

TagList

```
/*
* TagList.java
* LinkedList container for Tags.
*
* Author:   Alex Lindeman
* School:   Washington-Lee High School
* Modified: Mar 3 2010
*
* IDE:      gedit 2.28.0
* System:   IBM ThinkPad X31
*/

class TagList {

      // globals
      private TagNode head = null;
      private TagNode tail = null;
      private int size = 0;

      // constructor
      TagList (){ }

      // checks if list is empty
      public boolean isEmpty (){
            return size == 0;
      }

      // returns the size
      public int getSize (){
            return size;
      }

      // gets a single tag at a specific index
      public Tag getTag (int index){
            int count = 0;
            for(TagNode current = head; current != null; current = current.next){
                  if (count == index){
                        return current.tag;
                  }
                  count ++;
            }
            return null;
      }

      // gets all tags
      public void print(){
            int pos = 1;
            for(TagNode currentNode = head; currentNode != null; currentNode =
currentNode.next){
                  System.out.println("[" + pos + "] "+ currentNode.tag);
                  pos ++;
            }
```

```
	}

	// search method
	public int find(String searchterm){
		int count = 0;
		for (TagNode current = head; current != null; current = current.next){
			if (current.tag.name.equals(searchterm))
				return count;
			count ++;
		}
		return -1;
	}

	// appends a tag to the end of the list
	public void add (Tag newTag){
		TagNode temp = new TagNode();
		temp.tag = newTag;

		if (isEmpty()){
			head = temp;
			tail = temp;
		} else {
			tail.next = temp;
			tail = temp;
		}

		size ++;
	}

	// adds a tag to a specific index (don't really know why you'd want/need to,
but then again, why not)
	public void add (Tag newTag, int index){

		// adding to empty
		if (isEmpty()) {
			TagNode temp = new TagNode();
			temp.tag = newTag;
			head = temp;
			tail = temp;
			size ++;
		}

		// adding to beginning
		if (!isEmpty() && index == 0 )
		{
			TagNode temp = new TagNode();
			temp.tag = newTag;
			temp.next = head;
			head = temp;
			size ++;
		}

		// adding to end
		if (!isEmpty() && index == size)
		{
```

```
                TagNode temp = new TagNode();
                temp.tag = newTag;
                tail.next = temp;
                tail = temp;
                size ++;
          }

          // adding to middle
          if (!isEmpty() && index < size && index > 0){
                int count = 0;
                TagNode prev = head;
                for(TagNode current = head; current != null; current =
current.next){
                      if (count == index){
                            TagNode temp = new TagNode();
                            temp.tag = newTag;
                            temp.next = current;
                            prev.next = temp;
                      }
                      prev = current;
                      count ++;
                }
                size ++;
          }

          // adding to non-existent index
          if (!isEmpty() && index > size){
                System.out.println("Could not add tag " + newTag + " to index " +
                      index + " (out of bounds)");
          }
     }

     // removes a tag at a particular index
     public Tag remove (int index){

          // invalid index
          if (index >= size || index < 0) return null;

          // removing from empty list
          if (isEmpty()) return null;

          // removing last remaining tag
          if (index == 0 && index != size - 1){
                Tag temp = head.tag;
                head = null;
                tail = null;
                size --;
                return temp;
          }

          // removing first tag (head)
          if (index == 0 && head != tail){
                Tag temp = head.tag;
                head = head.next;
                size --;
```

```
                  return temp;
            }

            // removing last tag (tail)
            if (index == size - 1 && size > 1){
                  Tag temp = tail.tag;
                  for (TagNode current = head; current != null; current =
current.next){
                        if (current.next == tail){
                              tail = current;
                              tail.next = null;
                              size --;
                              return temp;
                        }
                  }
                  return temp;
            }

            // removing from middle
            if (index != 0 && index != size -1){
                  int count = 0;
                  TagNode prev = head;
                  for (TagNode current = head; current != null; current =
current.next){
                        if (count == index){
                              Tag temp = current.tag;
                              prev.next = current.next;
                              size --;
                              return temp;
                        }
                        prev = current;
                        count ++;
                  }
                  return null;
            }

            // if for some reason nothing happens, do nothing
            return null;
      }
}
```

TagNode

```
/*
* TagNode.java
* Class definition for Tag nodes (used in LinkedList).
*
* Author:   Alex Lindeman
* School:   Washington-Lee High School
* Modified: Mar 3 2010
*
* IDE:      gedit 2.28.0
* System:   IBM ThinkPad X31
*/

class TagNode {

      // globals
      public Tag tag = null;
      public TagNode next = null;

      // constructor
      TagNode (){ }

}
```

Tag

```
/*
* Tag.java
* Class definition for Tags.
*
* Author:   Alex Lindeman
* School:   Washington-Lee High School
* Modified: Mar 3 2010
*
* IDE:      gedit 2.28.0
* System:   IBM ThinkPad X31
*/

class Tag {

      //globals
      public String name;

      // construct
      Tag (){
      }

      Tag (String newName){
            name = newName;
      }

      // as string
      public String toString(){
            return name;
      }
}
```

C2. Usability

Feature	Documentation
Intuitive menus	Any screenshot
Minimal input	Any screenshot
Data manipulation through search results	Screenshot #6
Fast search method	Screenshot #5
Allow user to copy picture to the desktop	Screenshot #6
Exclusion of unnecessary features	Screenshot #1

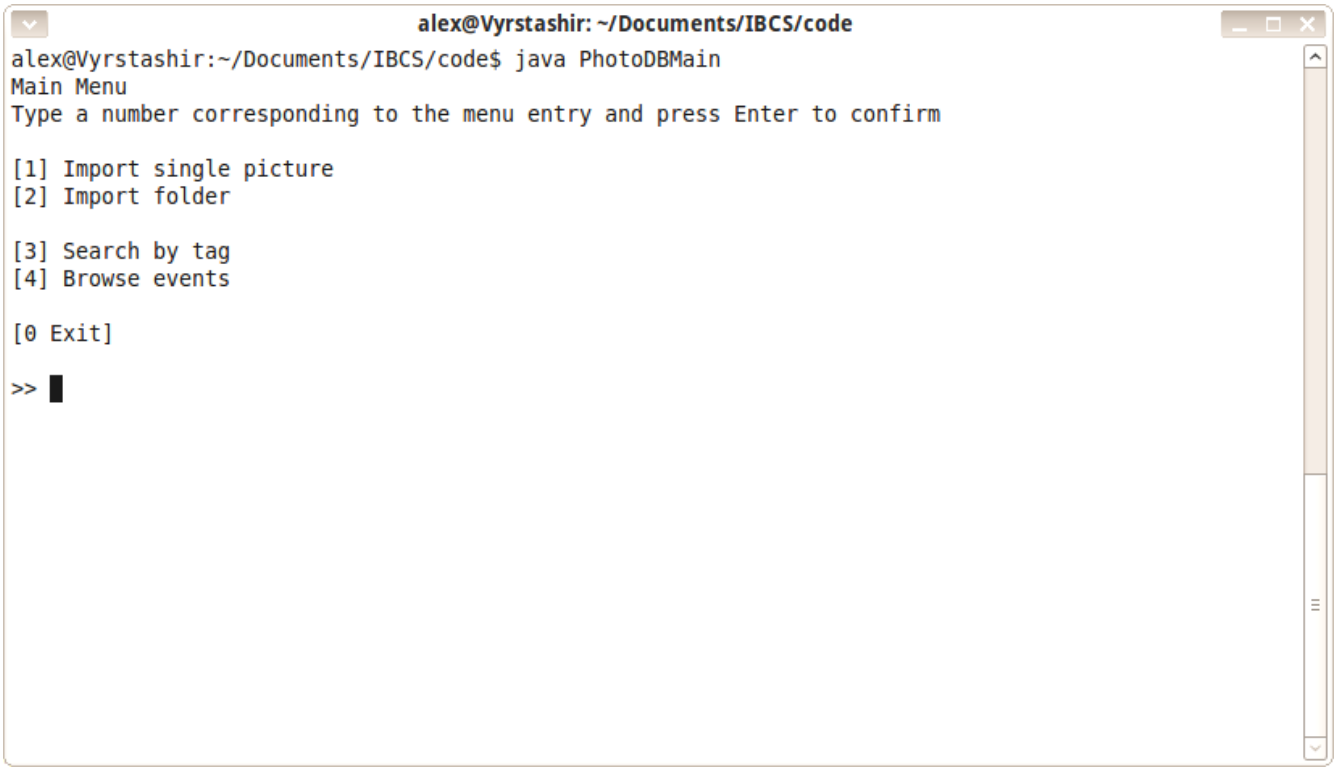
C3. Handling Errors

Error	Resolution	Location
User does not have filesystem permissions to import pictures or folders of pictures	Notifies the user that the folder is not accessible and aborts import	Page 18
User cannot copy the picture to the desktop	Notifies the user and aborts	Page 19
User inputs invalid name or search term	Asks user to repeat input	Page 16
User attempts to quit when they may not actually want to	Asks for verification	Page 22

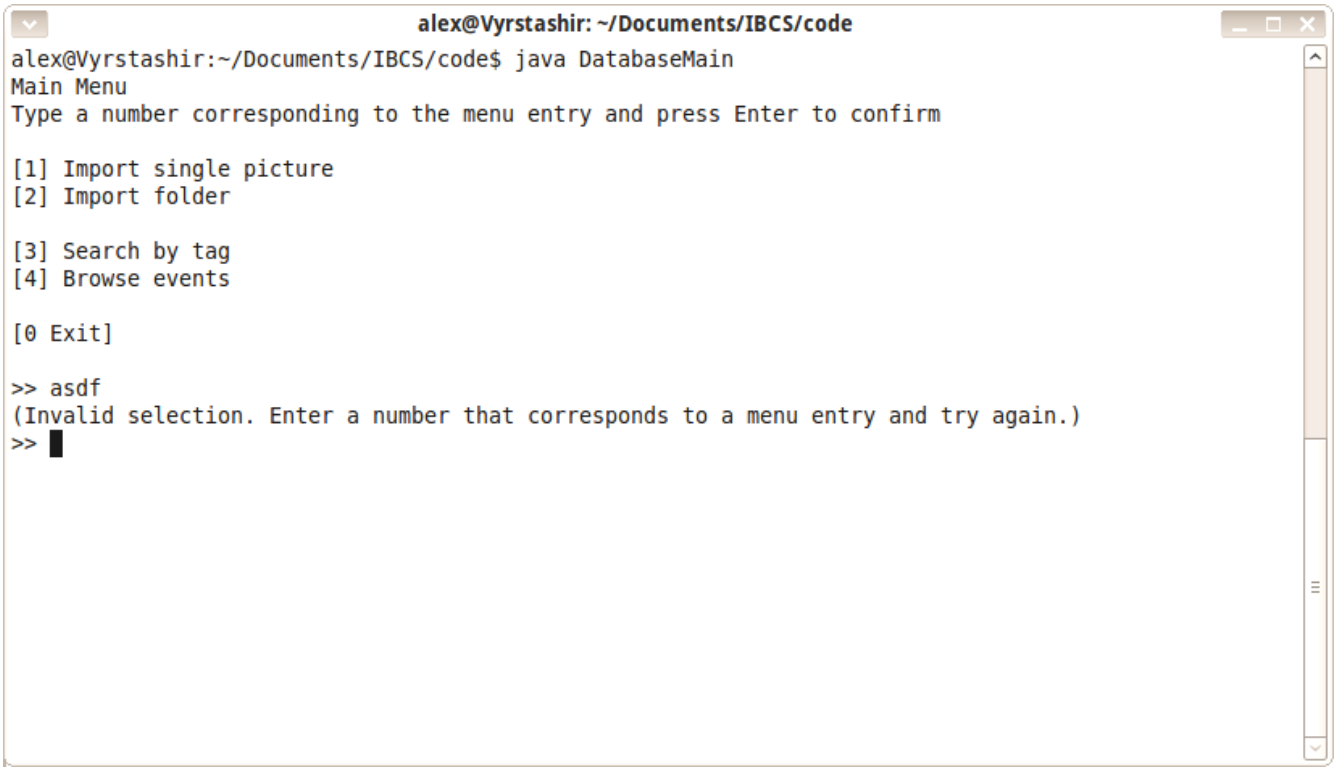
C4. Success of Program

Objective	**Evidence**
Easy to use menu system	Any screenshot
Minimal input	Any screenshot
Data manipulation through search results	Screenshot #6
Store pictures for the user intuitively	Screenshot #3
Allow the user to transfer a picture to the Desktop	Screenshot #6
Exclude unnecessary features	Screenshot #1

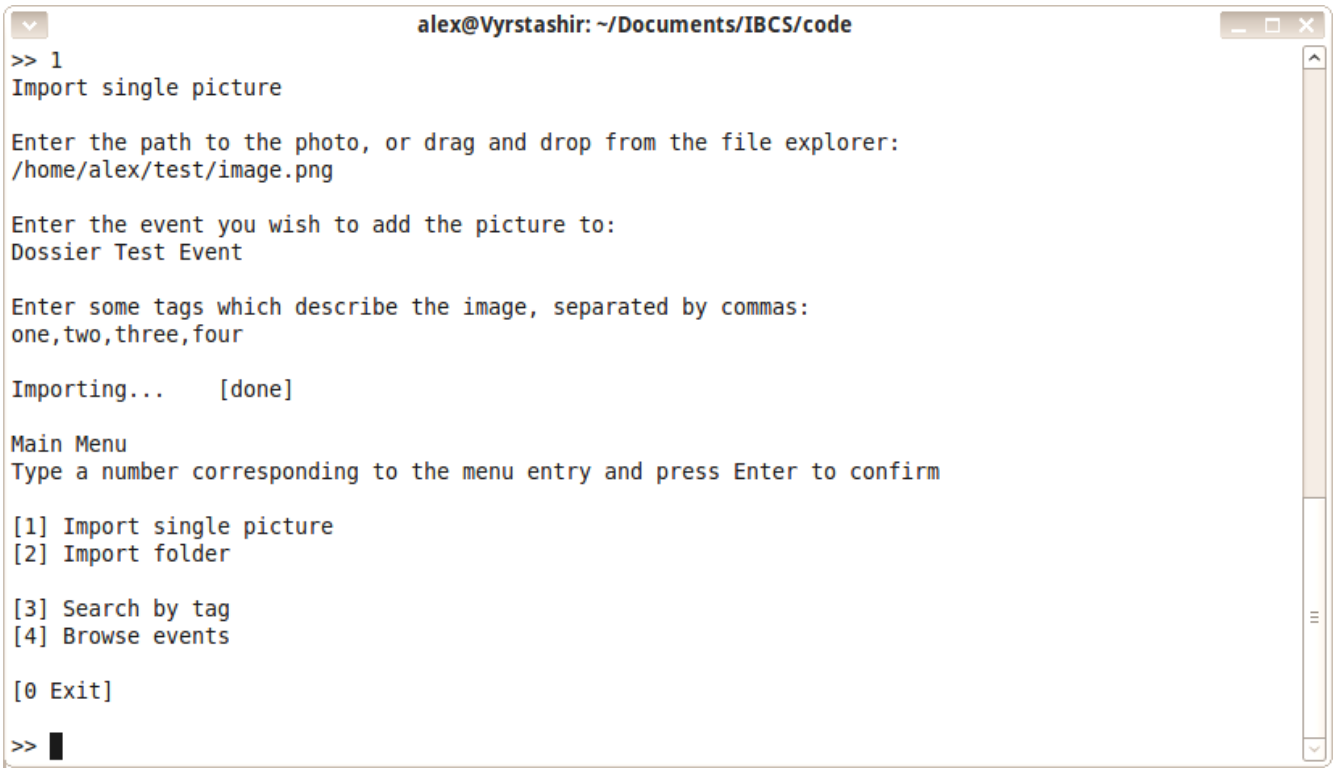
D1. Test Output



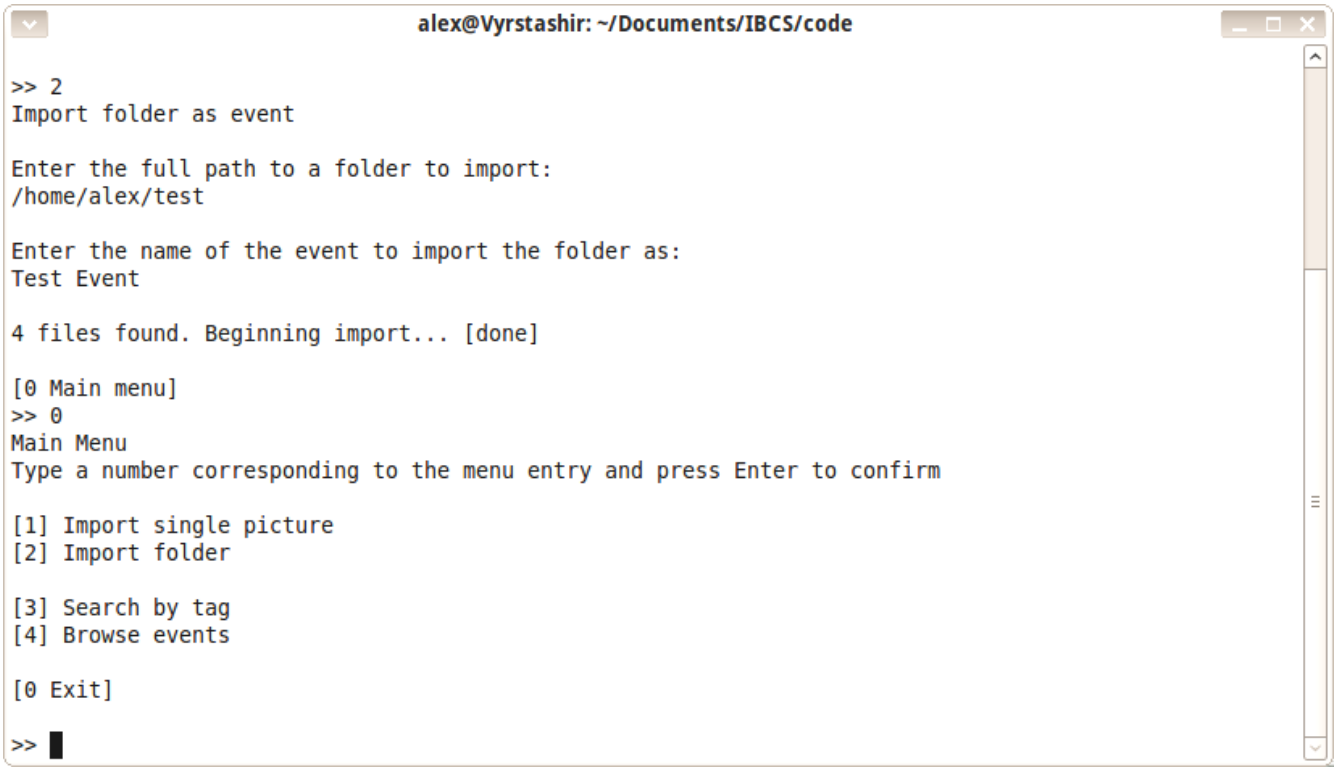
Screenshot 1: The main menu upon starting the program; illustrates the easy-to-use menu system.



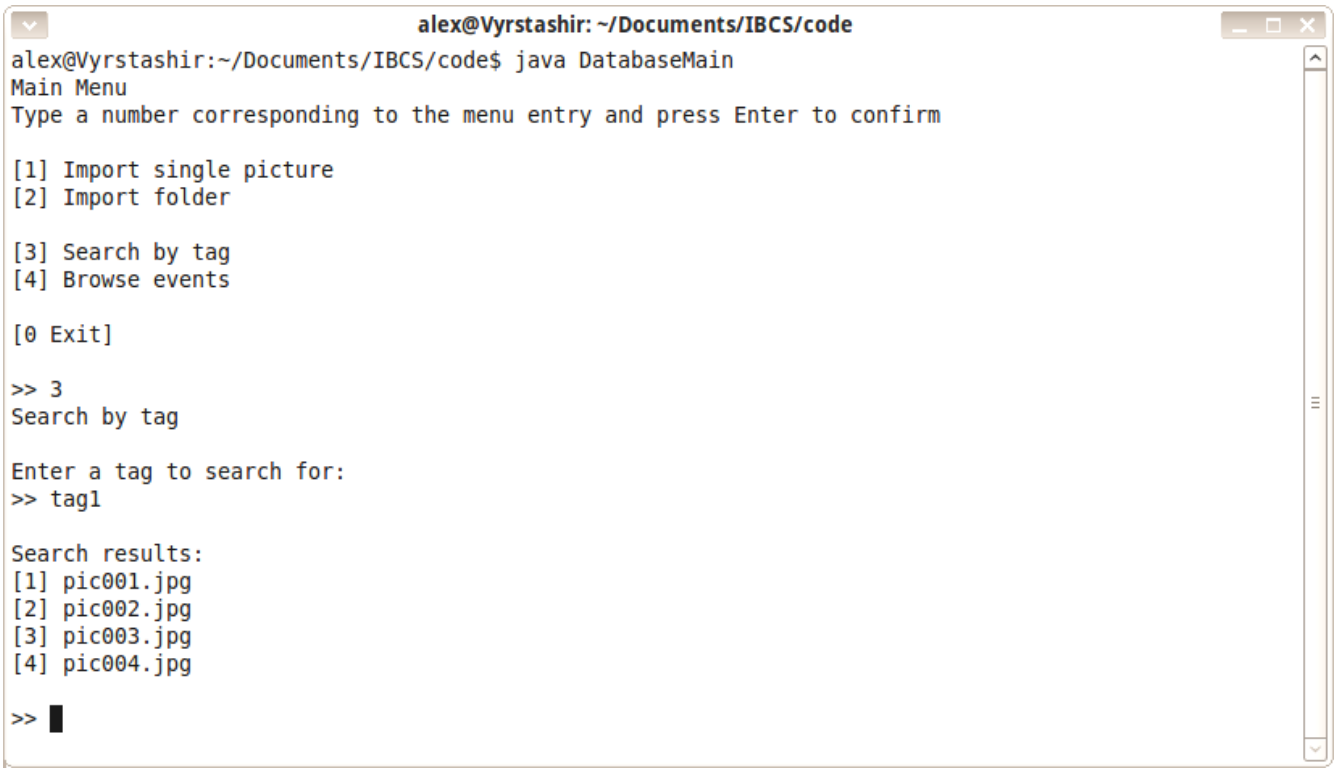
Screenshot 2: User attempts to input an invalid selection.



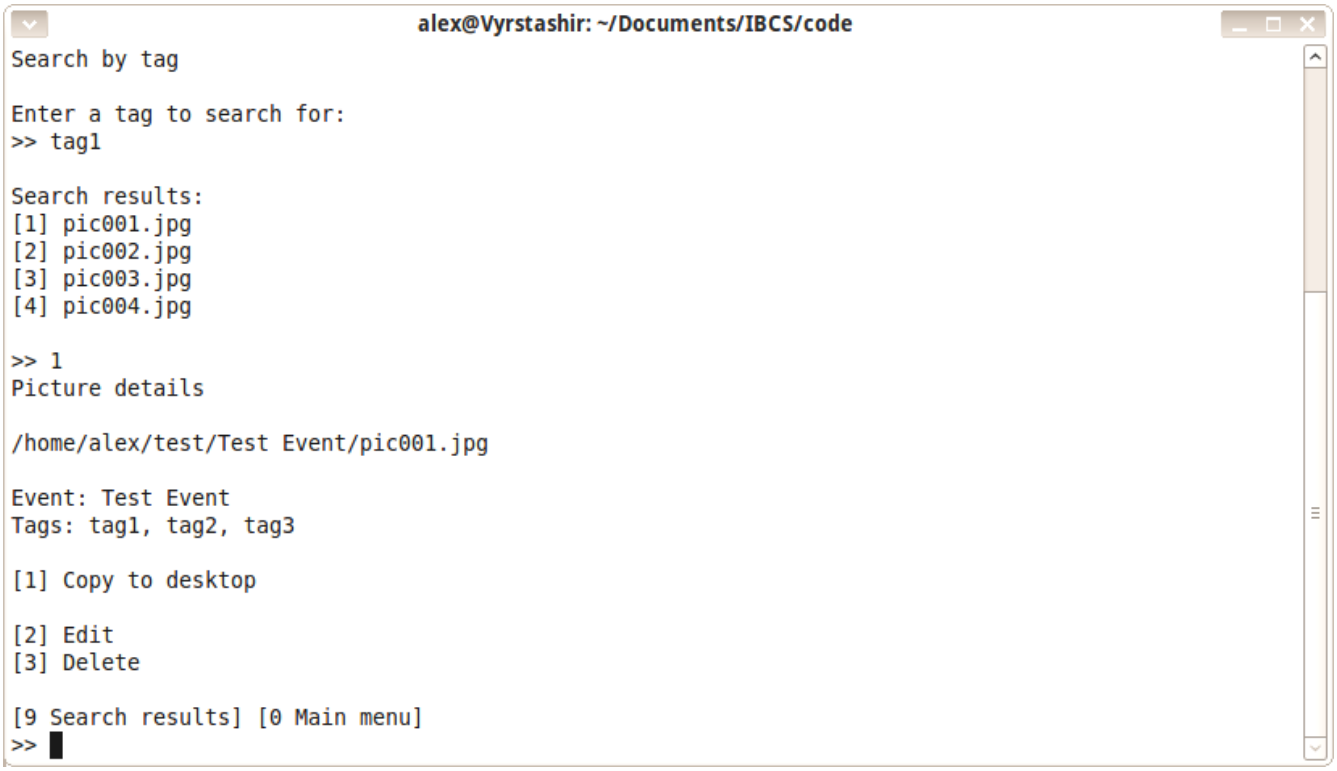
Screenshot 3: User imports a single picture from a file.



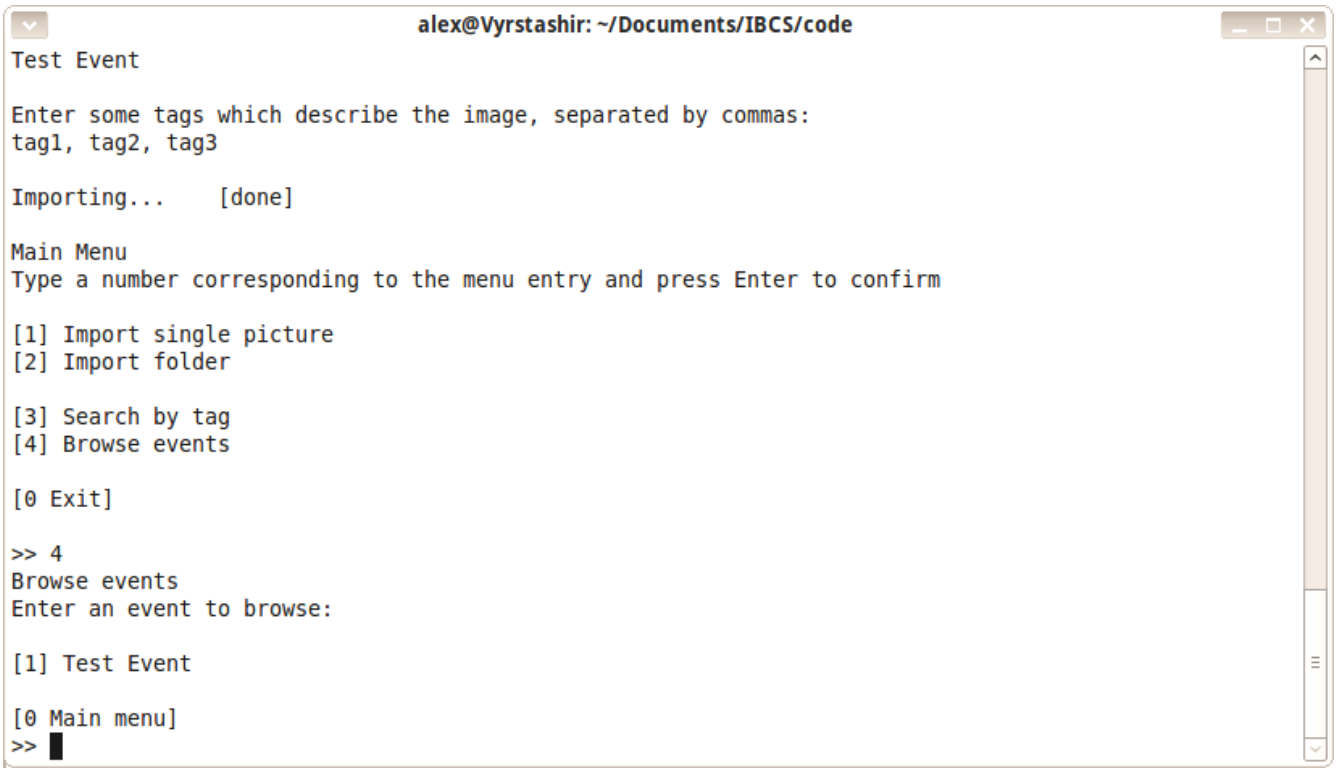
Screenshot 4: User imports a folder containing four pictures and adds them to a new event called "Test Event."



Screenshot 5: User searches for tag "tag1" and gets all pictures which match the results.



Screenshot 6: User views details of a single picture and possible actions for it.



Screenshot 7: User lists all events currently in the database (in this case, just one, called "Test Event")

D2. Solution Evaluation

Functionality

The solution worked well in the end. All of the objectives were met successfully, and it works for all data sets. The menu system was designed as simply as possible and is easy for people to understand, with minimal input. The searching system allows users to search through the database with ease, and copy pictures to the desktop. It is as simple as possible to eliminate clutter and unneeded “bloat” in the program.

Efficiency

The program is efficient in that it uses 3 LinkedLists which are linked together, but the searching method is very slow, as its Big O complexity is O(n). The search could be improved by sorting the LinkedLists every time new data is added, but this would further reduce the speed at which the program works; or by using a BinaryTree instead of a LinkedList, but this would have been difficult to implement.

Limitations

The program has many limitations. It is very basic, in that all it can do is ask for pictures, store them in folders, and search for them. It could have many other features, such as email and printing functionality, setting a picture as the desktop image, etc.. However, these changes would add bulk to the program, and due to operating system differences these features would be difficult to implement and would interfere with one of the objectives of the system.

Design

The initial design, a 3-dimensional LinkedList, was appropriate for the solution. 3 separate LinkedLists could have been used, but would have been inefficient. Text files to store the data probably could have been used but the time constraints of this project did not allow them to be implemented, and the method to store them also would have been difficult to figure out. The program also could have been designed to directly print or email pictures for the user, but due to individual system settings and differences it would have been impractically complicated to add to the program.

D3. User Manual

Installation

1. Copy the *database* folder to your home folder.
2. Open a terminal window by going to Applications, clicking Accessories, and clicking Terminal.
3. Type "`cd database`" into the terminal window.
4. Type "`javac DatabaseMain.java`" to create the database program.
5. When the program is finished creating the database, type "`java DatabaseMain`" to run the program. Your program is set up and from now on you can skip step 3 when running the database program.

Using the Menu System

- Whenever a menu appears, options will have numbers associated with them, in brackets to their left; for example:

```
[1] Import single picture
```

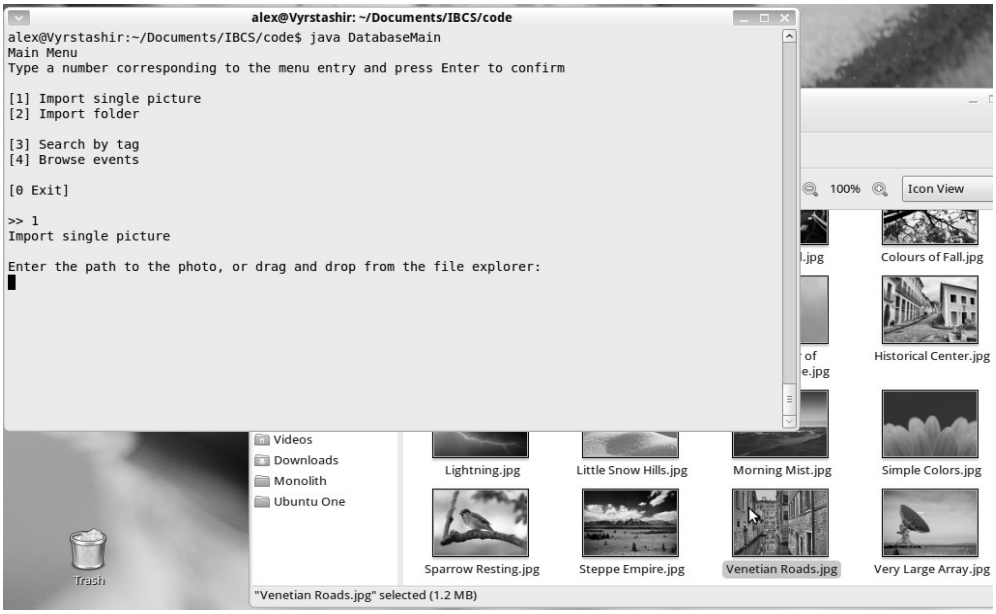
- At the bottom of the terminal window, there will be 2 carets followed by a blinking box or line (for example, >> _). Simply press a number on your keyboard followed by the Enter key to navigate through corresponding menu entries.

Importing Pictures

Single Picture

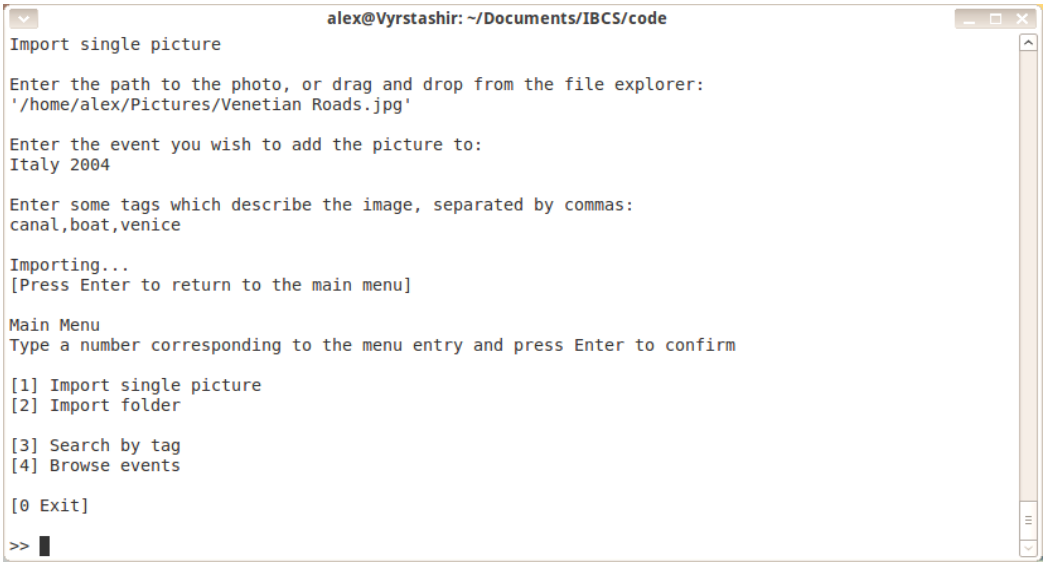
- Importing a single picture will add one picture to the photo database.

1. Open your file browser to the picture you wish to import.
2. From the main menu, select "Import single picture" by pressing 1 and then Enter.



3. The program will ask you for a path to the picture. Drag the picture in the file browser to the terminal window.
4. A folder path will be pasted into the window, for example, "`'/home/alex/Pictures/Lightning.jpg'`", and press Enter.
5. Enter an event to store the picture in, for example, "Florida 2010", and press Enter.
6. Enter a few tags to describe the image, such as "lightning, weather, storm, Florida", and press Enter.
7. Press the Enter key and the picture will be added to your database.

8. Press Enter again to return to the main menu.

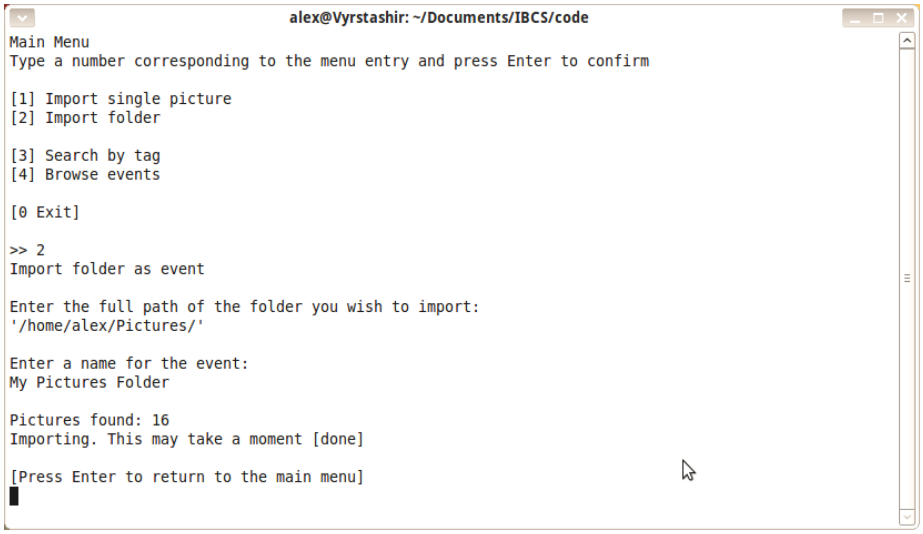


Folder of Pictures

- The program can add an entire folder that has been taken off of your camera and transferred to your computer, and transfer them into a new event in the database.

1. In the file browser, find the folder you wish to import into the database.
2. From the main menu, select “Import folder” and press Enter.

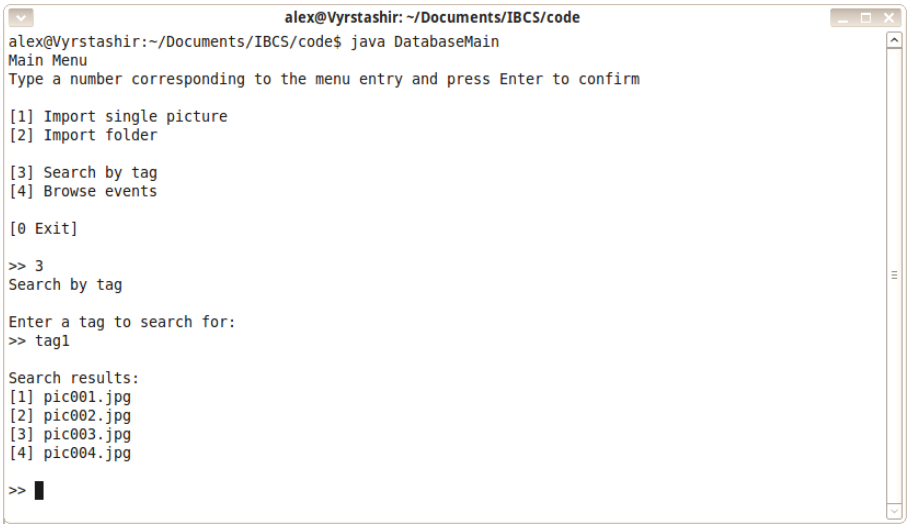
3. The program will ask you for a folder path. Drag and drop the folder onto the terminal window and press Enter.
4. Enter a name to call the event, like “California 2007”, and then press Enter.
5. The program will import all of your pictures, which could take a moment depending on the age of your computer. When it is done, press Enter to return to the main menu.



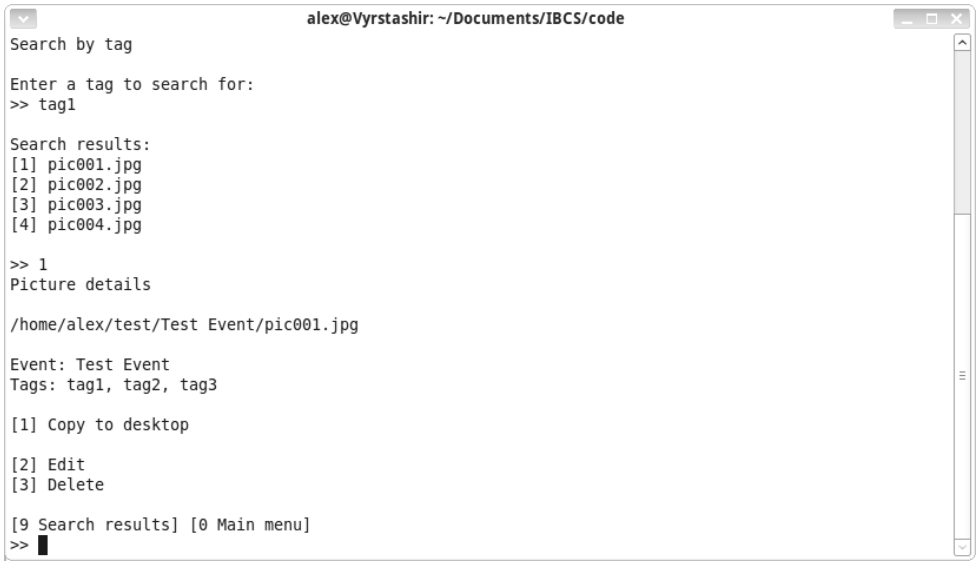
Searching for Pictures by Tag

- The program can search through the entire database through tags associated with pictures.

1. From the main menu, select “Search tags”.
2. Enter a tag you wish to search for, like “lightning”, and press Enter.



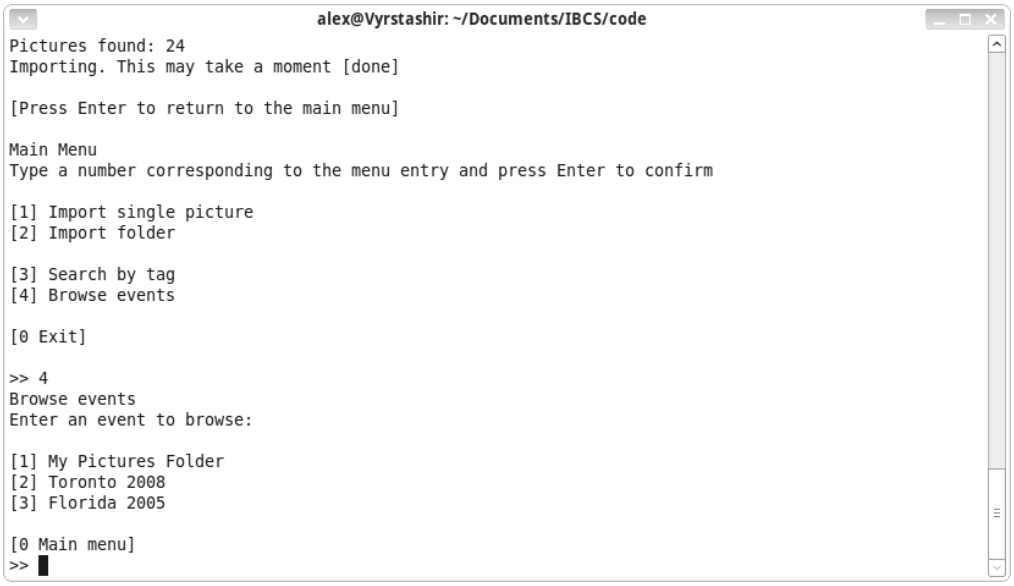
3. The program may take a moment to search, depending on the speed of your computer, and will display every picture that has the tag “lightning”.
4. Select a picture by entering its corresponding number and pressing Enter.
5. This will bring up a screen that displays all of the details of the picture. From this screen, you can copy the image to your desktop to quickly email it, print it, or edit it in another program; edit its details; or remove it entirely from the database.



Browsing Events

- The program can search through all of the events in the database and display the pictures within them.

1. From the main menu, select “Browse events”.
2. A list of all of the events stored in the database will appear. To browse through one, type its corresponding number and press Enter.



3. This will show all of the pictures within the event. From here, you can select a picture with its number and copy it to the desktop, edit its details, or delete it.

Mastery Aspects

#	Aspect	Page	How	Why
1	Encapsulation	36	The Picture class encapsulates two different data types (a String and TagList) into a single class definition.	Data organization into a single object is much more efficient than storing many single variables.
2	Parsing a text file or data stream	17-18	The program gets information about the contents of a folder to import several pictures at once.	Allows the user to insert many pictures into the database more easily than adding them one by one.
3	Implementation of a hierarchical data structure	38-43	The program has three data objects which are stored by each other and a main method which uses them all.	A hierarchical data structure is the most logical and efficient way to store data in a program.
4	Polymorphism	30, 36-37, 43	There are multiple constructors for all of the fundamental objects in the program.	Multiple constructors allow for temporary objects to be created quickly.
5	Implementation of abstract data types	25	The isEmpty() function in all three LinkedLists checks if the LinkedList is empty or not.	The function is required for manipulating the head and tail elements in the LinkedList.
6		26	The add() function is important to be able to add things to the LinkedList.	Without the ability to add things the data in the LinkedList would be difficult to change.
7		27	The remove() function is important to be able to take out things in the LinkedList.	Without the ability to remove things the data in the LinkedList would be difficult to change.
8		25-26	The print() function allows for the user to be able to see what is inside the LinkedList easily.	Without being able to print the LinkedList, the user would not be able to know what is inside it and be able to edit it.

9	Use of five SL mastery factors		*See below*	
	User-defined objects	36-37	A Picture class is used in the database.	A user-defined object is needed because of the multiple types of data that make logistical sense to be stored in a single place.
	Simple selection	18	An if statement determines whether or not a directory was successfully found.	An if statement is best used because there are few possibilities for the function and it all that is being checked is a simple integer.
	Complex selection	17-22	The main menu uses a switch/case function to determine what the user wants to do.	A switch/case function is best in this situation because of the limited data possibilities and simplicity of the function.
	Use of sentinels or flags	17, 22	The main menu is encased in an otherwise infinite loop until a boolean variable “exit” is made true.	A flag which tells the loop to exit is the easiest way to make an otherwise infinite loop run as many times as you want while still be under control.
	Searching	26	A find method finds the data the user wants by linearly traversing the LinkedList, finding matches, and printing them to the screen.	Searching the database is the quickest way to find data rather than the user browsing through every element in the database and using unnecessary amounts of time doing so.