Table of Contents

A1: Analysis of the Problem

Introduction

Organization is a key part of success for individuals and businesses alike. Good organization involves good time management, planning, keeping track of events and tasks, and having necessary tools and information easily accessible. An inability to meet these requirements can have detrimental side effects. Poor time management and planning can waste valuable time. The lack of control over plans may cause chronic tardiness. Not having easy access to the equipment and data needed to complete tasks can cause low productivity. The overall result of disorganization is stress and an inability to meet goals.

Problem

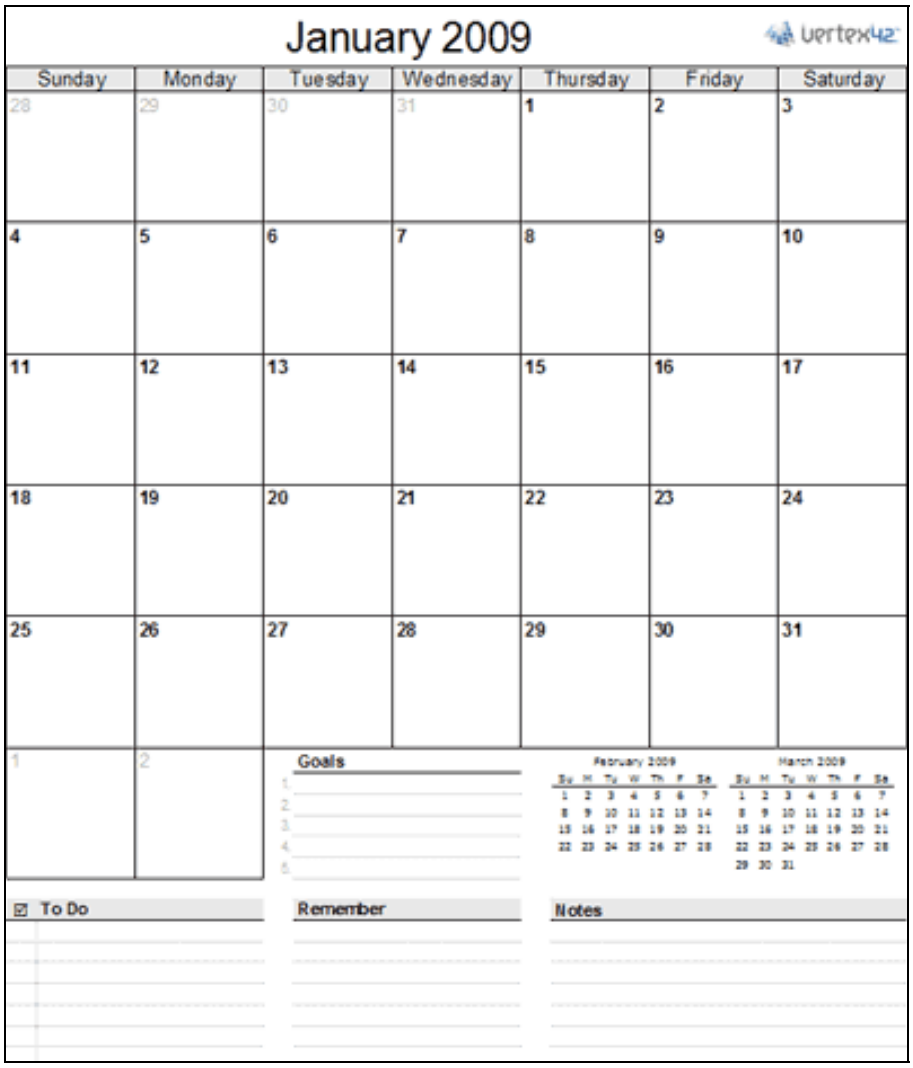
The underlying problem that causes disorganization is the misplacement of objects and facts. A small misplacement of a phone number or failure to recall important dates can morph into bigger losses. Scattering information, like a phone number, as noted above, includes wasting time in the search, the potential loss of a contact, and a sense of frustration and powerlessness. If such misplacement and incompetence can be reduced, disorganization may be eliminated.

A system of organization may increase punctuality, efficiency, and productivity. Individuals, ranging from students to professionals, could use a quick and simple way to keep track of their schedules. When all information, such as appointments, can be available, it becomes easier to gauge the amount of time available for new activities and work around important dates.

Past Solutions

In attempts to maintain order, people have turned to their memories, calendars, and planners. The use of memory fluctuates in effectiveness and does not work for everyone, as people have different capacities to recall important details. Generally, reliance on memory slows people down, especially when the amount needed to be noted increases. Calendars and planners work well to an extent. Calendars allow people to visualize their time, but only provide a snapshot, as multiple events and details usually cannot be recorded under the same day. Planners can provide more room and include calendars for snapshots of longer time periods. Both calendars and planners may be difficult to edit because changes require erasing, rewriting, and crossing out. If they are physically damaged or misplaced, the information contained in them may be very difficult to recover. Electronic calendars and planners exist, which reduces the chances of losing information and allows for changes to be made more easily.

A system of organization and keeping up with activities and events should take various details of the activity and time it will occur. The database should allow all aspects of an activity to be recorded and searched so they can be easily found in one location.



The above picture is an example of an existing paper planner or calendar which records dates and activities.

End-User

End-User: The end-user will be students and professionals who would like to use an electronic calendar database to keep track of events and activities.

End-User Questions

- What events or activities are you involved in?
- Who else is participating in the events and/or activities?
- How would you prioritize your activities in order of importance?
- When does your schedule encounter the most conflicts?
- What calculations do you need?
- Do you need to make changes to your schedule?
- Do you need file I/O?
- How far ahead do you need to plan for?

End-User Requirements for System

- Able to perform quick searches
- Able to edit and save information
- Searchable by attributes
- User friendly
- Able to track activities and times
- Can detect time conflicts

Input

- Activity/Event Name
- Month of Activity
- Date of Activity
- Time
- Location
- People Involved
- Notes/Description
- Priority Value (1 being least important, 5 being most important)

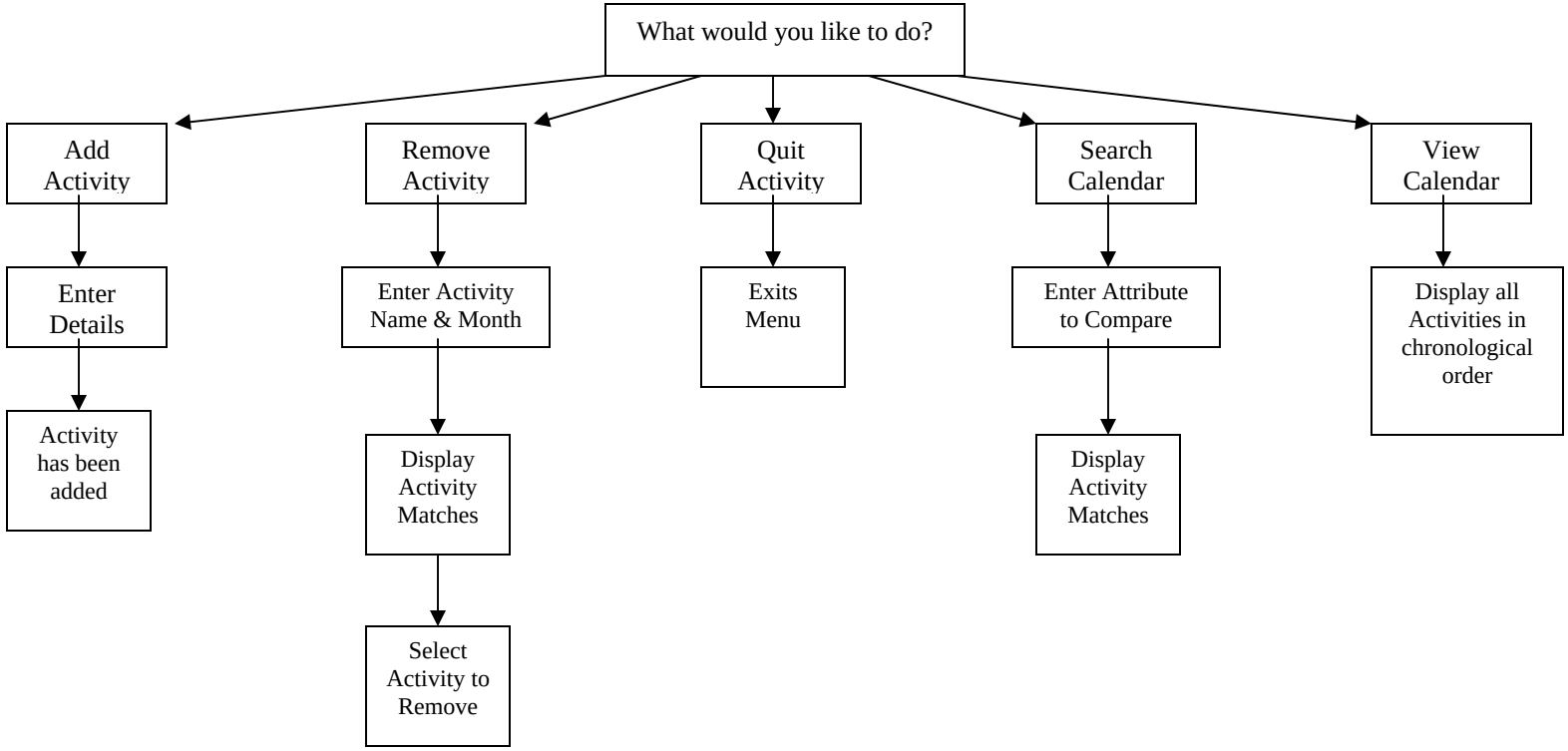
Output

- Activity Information (name, date, time, location, people, notes, etc)
- Conflicts within Activities
- Options to Resolve Potential Conflicts
- List of Activities that Match Search

Interfaces/Sub-Programs

- Display specific activity information (name, date, time, location, people, notes, etc)
- Display specific dates
- Search for activities by name, date, location
- Add or remove activities
- Edit/change information
- Normal user interface

User Action Flow Chart



A2: Criteria for Success

Introduction

This section will outline and list the objectives of an electronic calendar system and how it will function. It will describe how users will apply the functions of the system and database.

Behavior Outline

- Allow users to perform quick searches
- Allow users to add/remove entries
- Display previously entered data
- Search database using different categories (name, location, etc)
- Organize data chronologically
- Have user-friendly menus

Objectives

- Conduct quick searches
  - This will allow users to locate information efficiently, saving time and energy.
- Store and display data in chronological order
  - This will allow users to easily visualize and organize their time.
- User-friendly menus
  - This will allow users to easily manipulate the database, reducing stress.
- Easily add/remove
  - This will allow users to manipulate the calendar without ruining the aesthetic appeal or order of it.

Goals

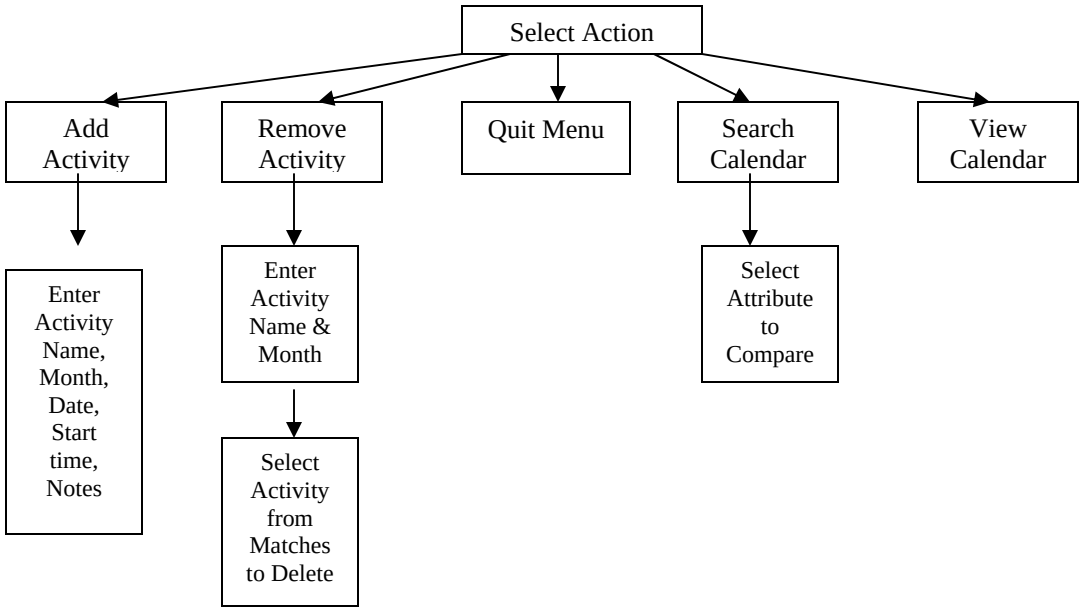
- Efficient searches
- Easy to follow interfaces
- Input error detection concerning dates and times
- File Input/Output

Environment Requirements/Operating Restrictions

- Can be run on PC
- Terminal
- Lots of hard drive space for searches and storing data
- Printers may be used to obtain hard copies of calendars

A3: Prototype Solution

Design/User-Action Flow Chart



Welcome!

What would you like to do?

a) Add Activity
b) Remove Activity
c) Search Calendar
d) View Calendar
e) Quit

Enter choice: a

Select Action

Add New Activity

Activity Name: Soccer Practice
Month: March
Date: 22
Start Time: 4:00 PM
Location: Quincy Field
Notes: _____

Add Activity

Welcome!

What would you like to do?

a) Add Activity
b) Remove Activity
c) Search Calendar
d) View Calendar
e) Quit

Enter choice: b

Select Action

Remove Activity

Please enter….
Activity Name: Soccer Practice
Month: March

Enter Activity Name & Month

Select Activity to Delete from Matches

Activity Matches….

1
Activity Name: Soccer
Month: March
Date: 22
Start Time: 4:00
Location: Quincy Field

2
Activity Name: Dinner
Month: March
Date: 25
Start Time: 5:00
Location: Restaurant

Enter choice:___

Select Activity from Matches to Delete

Welcome!

What would you like to do?

a) Add Activity
b) Remove Activity
c) Search Calendar
d) View Calendar
e) Quit

Enter choice: c

Select Action

Search:
Which attribute do you want to compare?
a) name
b) date
c) location
d)

Enter choice: __
Enter attribute:_____

Select Attribute to Compare

Welcome!

What would you like to do?

a) Add Activity
b) Remove Activity
c) Search Calendar
d) View Calendar

Enter choice: d

Calendar:

Activity Name: Soccer
Month: March
Date: 22
Start Time: 4:00
Location: Quincy Field

Activity Name: Dentist
Month: March
Date: 25
Start Time: 3:00
Location: Office

User Feedback

- Can arrow keys be used?
- Could be tedious to type in month
- Would users really want to see only month?
- Can there be scroll down menus?

B1: Data Structures

Activity

An activity will contain all information related to and an activity, including its name, time, location, and any extra notes regarding it.

Example:

String	Activity Name: Dentist
Int	Month: 4
Int	Date: 5
Int	Start Time hour: 8
Int	Start Time min: 05
String	Location: Dentist Office
String	Notes: Bring insurance information

This information can be used to compare in searches. The user can select an attribute to use to locate a specific activity. More activities may be added and previously entered activities may be removed as well.

Calendar

The calendar will be the main data structure that stores all activities. The activities will be stored in a linked list by date and binary tree by time. This will allow the activities to be easily sorted, compared, and searched.

Example:

January (1)
1
1:00 -Lunch
3:00 - New Years Picnic
2
6:00 - Run
3

This example shows three days of a month stored in Calendar.

Linked List

The activities will be stored in a linked list by date (month and day). When a search is being conducted or the data is being sorted, the linked list would be used to sort through the months and dates. The right day can be located, narrowing down the search for the activity.

The alternative to a linked list could be an array for each month and day. Arrays were avoided because they are static and it is easier to add data to linked lists.

Comparing Activities

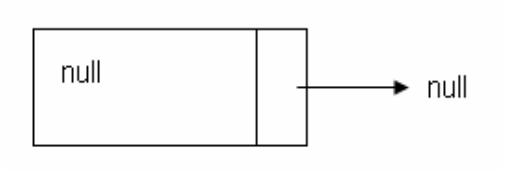
When a user wants to find an activity or remove an activity the user uses specific attributes of the activities to compare. The activities are stored in a linked list by date because it is easiest to use when everything must be accessed. It allows for repetition and is easy to remove from a linked list.

Comparisons can be made based on name, date, and location.

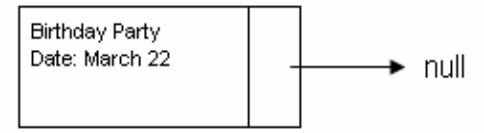
Activities are stored in a linked list for comparisons based on name, date, and location.

Example of Add to an Empty List:

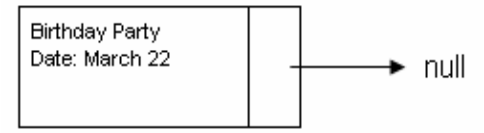
Before:



To be added:



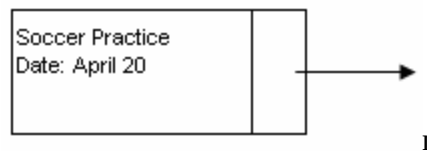
After:



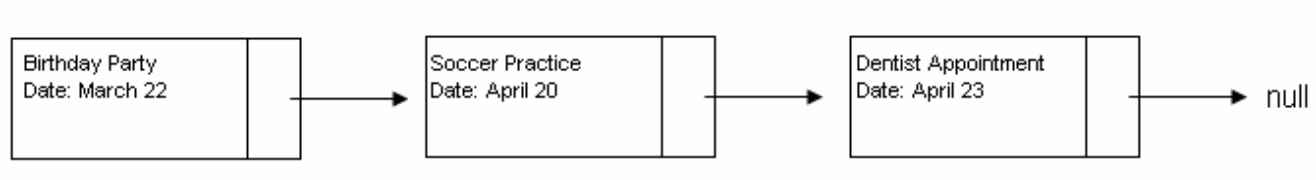
Example of Add to a non empty list:

Before:

To be added:

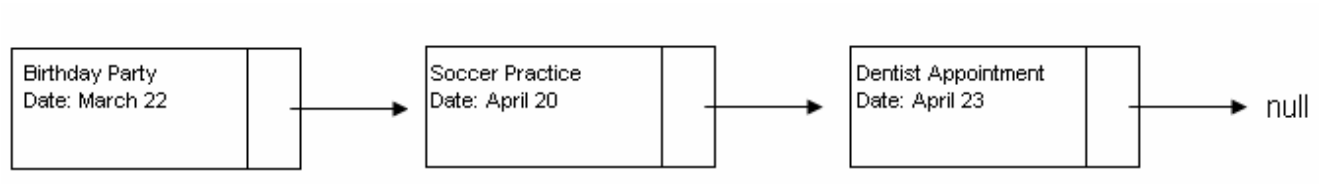


After:

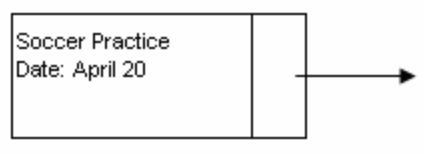


Example of Remove from a non empty list:

Before:



To be removed:



After:

B2: Algorithms

Name	add
Description	Adds an activity and its information into the Calendar.
Parameters	• String name • int month • int day • int startTime • int endTime; • String location • String notes
Return Value	none
Pre-Conditions	The calendar may or may not be empty.
Post-Conditions	A new activity has been added to the calendar
Pseudo code	1) Prompt user for attributes 2) Add activity to end of linked list 3) Once date match has been found, compare start times and end times of activities 4) If start time of new activity falls within start and end time of an activity already in the calendar, return both activities' information 5) Prompt user for his/her choice on how to resolve conflict (delete new activity, delete old activity, change start time of new activity)

Name	findName
Description	Finds matches based on the activity name.
Parameters	String name
Return Value	String toString
Pre-Conditions	The calendar is not empty.
Post-Conditions	Matches have been found based on name.
Pseudo code	1) Check to see if calendar is empty 2) Search through lists and activity names 3) Display list of all activities their corresponding information that match the name

Name	findLocation
Description	Finds matches based on the date the activity is scheduled for.
Parameters	String Location
Return Value	String toString
Pre-Conditions	Matches have been found based on date.
Post-Conditions	An activity has been removed from the calendar
Pseudo code	1) Check to see if calendar is empty 2) Search through lists and activity locations. 3) Display list of all activities their corresponding information that match the date.

Name	findDate
Description	Finds matches based on the date the activity is scheduled for.
Parameters	• int month • int day
Return Value	String toString
Pre-Conditions	Matches have been found based on date.
Post-Conditions	An activity has been removed from the calendar
Pseudo code	1) Check to see if calendar is empty 2) Search through lists and activity months and days. 3) Display list of all activities their corresponding information that date..

Name	print
Description	Edits information in an activity.
Parameters	none
Return Value	String toString
Pre-Conditions	The calendar is not empty.
Post-Conditions	Activities are displayed.
Pseudo code	1) Check to see if calendar is empty 2) Prompt user for month or all 3) Traverse through the lists for all activities that match the month and date, and return the activities' information.

Name	selectionSort
Description	Sorts activities in order by month and date.
Parameters	none
Return Value	void
Pre-Conditions	The calendar is not empty.
Post-Conditions	Activities are sorted by month, date, and start time.
Pseudo code	1) Check to see if calendar is empty 2) Compare months, if one is less than another, then switch them. 3) Traverse through the activities and compare dates, if one is less than another, then switch them.

Name	remove
Description	Removes an activity from the Calendar.
Parameters	String name
Return Value	none
Pre-Conditions	The calendar is not empty.
Post-Conditions	An activity has been removed from the calendar
Pseudo code	1) Check to see if calendar is empty 2) Prompts user for activity name 3) Searches through the list for activity name 4) Once a match has been found, the activity is removed from the list

B3: Modular Organization

⇒ class CalendarMain
⇒ **public static void remove**
- class Calendar
  - class ActivityLL
    - class ActivityNode
      - class Activity

⇒ class CalendarMain
⇒ **public static void add**
- class Calendar
  - class ActivityLL
    - class ActivityNode
      - class Activity

⇒ class CalendarMain
⇒ **public static void print**
- class Calendar
  - class ActivityLL
    - class ActivityNode
      - class Activity

⇒ class CalendarMain
⇒ **public static void find**
- class Calendar
  - class ActivityLL
    - class ActivityNode
      - class Activity

class CalendarMain: The CalendarMain consists of a menu, which asks the user which function she/he would like to run. They may add an activity, find an activity, remove an activity, change an activity, or print activities. Once a choice has been made, it goes to the submenu of the function the user chose.

- static void add: This method adds activities to the ActivityLL. It prompts the user to for attributes (i.e. name, location, date, month, etc). The Activity is then added to the ActivityLL.
- static void find: This method searches through the ActivityLL to find an activity. It prompts the user for known information (i.e. date and month, location, name, etc) and traverses through ActivityLL to find matches.
- static void remove: This method removes activities from the ActivityLL. It prompts the user an activity name, searches through the ActivityLL for the matching activity, then removes the first matching activity from the linked list.
- static void print:: The print method prints out all of the activities in ActivityLL.

class ActivityLL: The ActivityLL is a linked list that holds all of the entered activities. A linked list is the easiest structure because it allows for dynamic size.

- isEmpty: Checks to see of the ActivityLL is empty (empty of size = 0)
- add: Adds an ActivityNode to the end of the list
- remove: Removes an ActivityNode from the list
- selectionSort: Sorts the ActivityNodes in order of month, date, and startTime
- find: Compares attributes of all Activities and returns matches
- print: Prints all ActivityNodes in order of month, date, and StartTime

class Activity: The Activity class is used to create and store an activity. It takes in different data (i.e. month, date, startTime, location, name, notes) and has methods to "get" and "set" the data. The "get" method accesses the information in the activity and is used for comparisons. The "set" method is used to set or change information in the activity.

- get: Returns the different attributes as strings and integers
- set: sets different attributes
    - activity.setLocation("Little Theater);
- toString: Takes all of the attributes and puts them into one string for printing
- print: prints all of the attributes

class ActivityNode: Creates anode that stores an Activity and points to the next Activity in ActivityLL

class CalendarL: The Calendar stores the linked list of activities and allows all of the functions to be used.

- static void find: This method sesarches through the ActivityLL to look at the information for each Activity. It can find any activity using different attributes attribute.
- static void add: The add method is used to add activities to the ActivityLL.
- static void remove: The remove method is used to remove activities from the ActivityLL.
- static void print: The print method prints out all of the activities in ActivityLL or just one activity, depending on the user's choice.

C1: Code & Good Programming Style

AvtivityNode.java

```
/*
File Name: ActivityNode.java
Author: Suchana Costa
Date: February 2, 2010
School: Washington-Lee High School
Computer Used: Dell Optiplex GX620
IDE: GEdit
Purpose: Creates the Activity Node
*/


//ActivityNode Class

public class ActivityNode
{
	public Activity myActivity = null;
	public ActivityNode next = null;

	// default constructor
	public void ActivitytNode()
	{
	}
}
```

Activity.java

```
/*
File Name: Activity.java
Author: Suchana Costa
Date: February 2, 2010
School: Washington-Lee High School
Computer Used: Dell Optiplex GX620
IDE: GEdit
Purpose: Defines basic attributes for activity class
*/


public class Activity
{
	//attributes

	private String name = "";
	private int month = 1;
	private int date = 1;
	private int startTimeh = 0;
	private int startTimem = 0;
	private String location = "";
	private String notes = "";
```

```
	public Activity()
	{
	}

	public Activity(String newName, int newMonth, int newDate, int newStartTimeh, int newStartTimem,
String newLocation, String newNotes)
	{
		name = newName;
		month = newMonth;
		date = newDate;
		startTimeh = newStartTimeh;
		startTimem = newStartTimem;
		location = newLocation;
		notes = newNotes;
	}

	//setters (mutators)
	public void setActivityName(String newName)
	{
		name = newName;
	}

	public void setMonth(int newMonth)
	{
		month = newMonth;
	}

	public void setDate(int newDate)
	{
		date = newDate;
	}

	public void setStartTimeh(int newST)
	{
		startTimeh = newST;
	}

		public void setStartTimem (int newST)
	{
		startTimem = newST;
	}


	public void setLocation(String newL)
	{
		location = newL;
	}

	public void setNotes(String newNotes)
	{
		notes = newNotes;
	}

	//Accessors (getters)

	public String getActivityName()
```

```
	{
		return name;
	}

	public int getMonth()
	{
		return month;
	}

	public int getDate()
	{
		return date;
	}

	public int getStartTimeh()
	{
		return startTimeh;
	}

	public int getStartTimem()
	{
		return startTimem;
	}

	public String getLocation()
	{
		return location;
	}

	public String getNotes()
	{
		return notes;
	}
	public String toString()
	{
		 return name + " is scheduled on " + month + "/" + date + ", " + "at hour:" + startTimeh
+ " and minute:" + startTimem + ", at " +      location + ". " + "\n" + "Things to remember:" + "" + notes +
"\n"+ "\n";
	}

}
```

ActivityLL.java

```
/*
File Name: ActivityLL.java
Author: Suchana Costa
Date: February 2, 2010
School: Washington-Lee High School
Computer Used: Dell Optiplex GX620
IDE: GEdit
Purpose: Creates a Linked List to store data
*/


public class ActivityLL
{
	//attributes
	private ActivityNode head = null;
	private ActivityNode tail = null;
	private int size = 0;

	//default constructor
	public ActivityLL()
	{
	}

	// isEmpty() returns true if the LinkedList is empty, and false otherwise
	public boolean isEmpty()
	{
		return size == 0;
		//return head == null;
	}

	//Add an Activity to the LinkedList (to the end of the list, not paying attention to order)
	public boolean add(Activity newActivity)
	{
		if (isEmpty())
		{
			ActivityNode myNode = new ActivityNode();
			myNode.myActivity = newActivity;
			head = myNode;
			tail = myNode;
			size++;
			return true;
		}

		//Add an Activity to the LinkedList (to a nonempty list)
		if (!isEmpty())
		{
			ActivityNode myNode = new ActivityNode();
			myNode.myActivity = newActivity;
			tail.next = myNode;
			tail = myNode;
			size++;
			return true;
		}
```

```
		return false;
	}


	//Remove an Activity
	public void Activity remove(String name)
	{
			//Cannot remove from an empty list
			if (isEmpty())
			{
				return null;
			}

			//Remove an activity from a nonempty list
			if (!isEmpty())
			{
				ActivityNode previous = head;

				for (ActivityNode current=head; current !=null; current= current.next)
				{

					if(current.myActivity.getActivityName().equals(name))
					  {
						//Set the previous node to the node after the current
and decrease size
							previous.next = current.next;
							size --;
					  }

					  previous = current;
				}

			}
		}

	///Find Activity by date
	public int findDate(int date, int month)
	{

		int count = 0;

		for (ActivityNode current=head; current != null; current= current.next)
		{
				//if the date and the month match, return the index (count)
			if(current.myActivity.getDate() == date && current.myActivity.getMonth()==
month)
			{
				//System.out.print(current.myContact);
				return count;
				}

				count ++;

		}

		return -1;
```

```
 }


//Find Activity by name by recursion
private int findName(String name, ActivityNode node, int count)
{
	if (node == null)
		return -1;
	if (node.myActivity.getActivityName().equals(name))
		return count;
	return findName(name, node.next, count + 1);
}



//Find Activity by location
public int findLocation(String location)
{

	int count = 0;

	 for (ActivityNode current=head; current !=null; current= current.next)
	 {

		 if(current.myActivity.getLocation().equals(location))
		{

			//System.out.print(current.myContact);
			return count;
		}

		count ++;
	 }

	return -1;
}

//Get the Activity you had searched for or get Activity at specific index
public Activity getActivity(int index)
{
	int count = 0;

	for(ActivityNode current = head; current !=null; current = current.next)
	{
		//if you are at the correct current, set the previous next to the current's nex
		if (count == index)
		{
			return current.myActivity;
		}

		count ++;
	}

	return null;
 }
```

```
	//get size
	public int size()
	{
		return size;
	}


	//print whole calendar (traverses the linked list and prints out the Activity info (date, time, etc) in
each NOde
	public void print()
	{
		for (ActivityNode current = head; current != null; current = current.next)
		{
			System.out.print(current.myActivity);
		}
	}

	public void selectionSort()
	{
		ActivityNode minNode;
		for (ActivityNode front = head; front != null; front = front.next)
		{
			minNode = front;
			for (ActivityNode current = front; current !=null; current = current.next)
			{
				//Compare months to see which is the earlier occurying month
				if ((minNode.myActivity.getMonth()<(current.myActivity.getMonth())))
				{
					//Compare dates to find the earlier occuring date
					if(minNode.myActivity.getDate() <
current.myActivity.getDate())
					{
						minNode = current;
					}

				}
			}
		//swap
		Activity temp = front.myActivity;
		front.myActivity = minNode.myActivity;
		minNode.myActivity = temp;
		}
	}

//to file
public void save(String filename)
   {
      //open file using filename
      RandomAccessFile file = new RandomAccessFile(filename, "rw");

      //write out to file the size
      Int saveSize = size;
      file.writeUTF(saveSize);


      //traverse through LL and writeUTF
      for (ActivityNode myNode = Cal.getHead(); myNode != null; myNode = myNode.next)
```

```
        {
            file.writeUTF(myNode.myActivity.getActivityName());
            file.writeUTF(myNode.myActivity.getMonth());
            file.writeUTF(myNode.myActivity.getDate());
            file.writeUTF(myNode.myActivity.getStartTimeh());
            file.writeUTF(myNode.myActivity.getStartTimem());
            file.writeUTF(myNode.myActivity.getLocation());
            file.writeUTF(myNode.myActivity.getNotes());
        }


        file.close();

        System.out.println("File saved under name " + filename);
    }

    public void load(String filename)
    {
        RandomAccessFile rfile = new RandomAccessFile(filename, "r");
        //Notice the "r", opening the file for reading

        String stringSizse = file.readUTF();
        //Read the first string that was originally written to file

        int inSize = Integer.parseInt(stringSize);


        for (int x = 1; x <= intSize; x +=1)
        {

            newActivity = new Activity();
            String loadName = file.readUTF(myNode.myActivity.getActivityName());
            String loadMonth = file.readUTF(myNode.myActivity.getMonth());
            int loadDate = file.readUTF(myNode.myActivity.getDate());
            int loadStartTimeh = file.readUTF(myNode.myActivity.getStartTimeh());
            int loadStartTimem = file.readUTF(myNode.myActivity.getStartTimeh());
            String loadLocation = file.readUTF(myNode.myActivity.getLocation());
            String loadNotes = file.readUTF(myNode.myActivity.getNotes());

            newActivity.setName(loadName);
            newActivity.setMonth(loadMonth);
            newActivity.setDate(loadDate);
            newActivity.setStartTimeh(loadStartTimeh);
            newActivity.setStartTimem(loadStartTimem);
            newActivity.setLocation(loadLocation);
            newActivity.setNotes(loadNotes);

            add(newActivity);
        }

        file.close();
        System.out.println("File has been loaded");
    }
```

CalendarL.java

```
/*
File Name: CalendarL.java
Author: Suchana Costa
Date: February 2, 2010
School: Washington-Lee High School
Computer Used: Dell Optiplex GX620
IDE: GEdit
Purpose: Defines functions for the Database 'Calendar'
*/

public class CalendarL
{
	// attributes
	private ActivityLL myActivityList = new ActivityLL();
	private int size = 0;

	// default constructor
	public CalendarL()
	{
	}

	//setters
	public void setMyActivityArray(ActivityLL newMyActivityList)
	{
		myActivityList = newMyActivityList;

	}

	//sets size
	public void setSize(int newSize)
	{
		size = newSize;
	}

	//getters
	public ActivityLL getMyActivityList()
	{
		return myActivityList;
	}

	public int size()
	{
		return myActivityList.size();
	}

	public void add(Activity newActivity)
	{
		myActivityList.add(newActivity);
	}

	public void print()
	{
		myActivityList.print();
```

```
}

public void selectionSort()
{
	myActivityList.selectionSort();
}

//Find Activity from name
public Activity findName(String name)
{
	int result = myActivityList.find(name);
	//check if found
	if (result != -1)
	{
		return myActivityList.getActivity(result);
	}

	return null;
}

//Find Activity from date
public Activity findDate(int d, int m)
{
	int result = myActivityList.find(m, d);
	//check if found
	if (result != -1)
	{
		return myActivityList.getActivity(result);
	}

	return null;
}

//Find Activity from location
public Activity findLocation(String location)
{
	int result = myActivityList.find(location);
	//check if found
	if (result != -1)
	{
		return myActivityList.getActivity(result);
	}

	return null;
}


//Find Activity by date
public String findDate(int date)
{
	int count = 0;

	//Traverses the LL for the first matching activity by date
	for (ActivityNode current=head; current !=null; current= current.next)
	{
		if(current.myActivity.getDate() == date)
		{
```

```
                        //System.out.print(current.myContact);
                        return current.myActivity.toString();
                }
        }

        return null;

}

//Find Activity by name
public String findName(String name)
{
        int count = 0;

        //Traverses the LL to find the first match by name
        for (ActivityNode current=head; current !=null; current= current.next)
        {
                if(current.myActivity.getActivityName().equals(name))
                {
                        //System.out.print(current.myContact);
                        return current.myActivity.toString();
                }
        }

        return null;
}


//Find Activity by location
public String findLocation(String location)
{
        int count = 0;

        //Traverses through the LL to find the first match of location
        for (ActivityNode current=head; current !=null; current= current.next)
        {
                if(current.myActivity.getLocation().equals(location))
                {
                        //System.out.print(current.myContact);
                        return current.myActivity.toString();
                }
        }

        return null;
}

public void print()
{
        //Traverses through the LL and prints out each Activity
        for (ActivityNode current = head; current != null; current = current.next)
        {
                System.out.print(current.myActivity);
        }
}
}
```

CalendarMain.java

```
/*
File Name: CalendarMain.java
Author: Suchana Costa
Date: February 2, 2010
School: Washington-Lee High School
Computer Used: Dell Optiplex GX620
IDE: GEdit
Purpose: Provides a menu for users to choose actions from
*/

import java.io.*;

public class CalendarMain
{
	private static BufferedReader stdin =
      new BufferedReader( new InputStreamReader( System.in ) );

	public static void main ( String [] args ) throws IOException
	{
		ActivityLL Cal = new ActivityLL();

		for(;;)
		{
			Activity a = new Activity();
			//Prints out the main menu for users to choose action
			System.out.println();
			System.out.println("The Calendar is Open");
			System.out.println();
			System.out.println("Select what you would like to do:");
			System.out.println();
			System.out.println("(a)dd activity");
			System.out.println("(r)emove activity");
			System.out.println("(f)ind activity");
			System.out.println("(p)rint Calendar");
			System.out.println("(q)quit");
			System.out.println();

			System.out.println();

			System.out.print("Enter choice:");
			String answer = stdin.readLine();


			if (answer.equals("a"))
```

```
{
        System.out.println("User selected ADD");
        System.out.println();

        System.out.print("Enter Activity Name:");
        String name = stdin.readLine();
        a.setActivityName(name);

        //Checks to see if the input was an integer, and if not,
prompts the user to reenter teh data
        int month = getCheckedInt("Please enter month (example:
If month is December, enter 12): ");

        //Checks to make sure month is within the acceptable range
of 0 to 13
        while (true)
        {
            while (month <= 0 || month >= 13)
            {

                int month1 = getCheckedInt("Please enter
month (example: If month is December, enter 12): ");
                month = month1;
            }

            //The following if statements print out what the user
has selected and asks the user to verify whether or not their choice is correct
            //If the user made a mistakem, the user may reenter
the month
            if ( month == 1)
            {
                System.out.println("You chose January");
                System.out.println("Is this correct? (y/n)");
                String m1 = stdin.readLine();

                if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                {
                    System.out.println("Month =>
January");

                    break;
                }

                else
                {
                    month = 13;
```

```
                    continue;

                }


            }


            else if ( month == 2)
            {
                System.out.println("You chose February");
                System.out.println("Is this correct? (y/n)");
                String m1 = stdin.readLine();

                if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                {
                    System.out.println("Month =>
February");
                    break;
                }

                else
                {
                    month = 13;
                    continue;

                }

            }

            else if ( month == 3)
            {
                System.out.println("You chose March");
                System.out.println("Is this correct? (y/n)");
                String m1 = stdin.readLine();

                if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                {
                    System.out.println("Month =>
March");
                    break;
                }

                else
```

```
                {
                        month = 13;
                        continue;

                }

        }

        else if ( month == 4)
        {
                System.out.println("You chose April");
                System.out.println("Is this correct? (y/n)");
                String m1 = stdin.readLine();

                if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                {
                        System.out.println("Month =>
April");
                        break;
                }

                else
                {
                        month = 13;
                        continue;

                }
        }

        else if ( month == 5)
        {
                System.out.println("You chose May");
                System.out.println("Is this correct? (y/n)");
                String m1 = stdin.readLine();

                if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                {
                        System.out.println("Month =>
May");
                        break;
                }

                else
                {
```

```
                                        month = 13;
                                        continue;

                                }
                        }

                        else if ( month == 6)
                        {
                                System.out.println("You chose June.");
                                System.out.println("Is this correct? (y/n)");

                                String m1 = stdin.readLine();

                                if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                                {
                                        System.out.println("Month =>
June");
                                        break;
                                }

                                else
                                {
                                        month = 13;
                                        continue;

                                }

                        }

                        else if ( month == 7)
                        {
                                System.out.println("You chose July");
                                System.out.println("Is this correct? (y/n)");

                                String m1 = stdin.readLine();

                                if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                                {
                                        System.out.println("Month =>
July");
                                        break;
                                }

                                else
```

```
                {
                        month = 13;
                        continue;

                }

        }

        else if ( month == 8)
        {
                System.out.println("You chose August");
                System.out.println("Is this correct? (y/n)");

                String m1 = stdin.readLine();

                if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                {
                        System.out.println("Month =>
August");

                        break;
                }

                else
                {
                        month = 13;
                        continue;

                }

        }

        else if ( month == 9)
        {
                System.out.println("You chose September");
                System.out.println("Is this correct? (y/n)");

                String m1 = stdin.readLine();

                if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                {
                        System.out.println("Month =>
September");

                        break;
                }
```

```
                    else
                    {
                        month = 13;
                        continue;

                    }

                }

                else if ( month == 10)
                {
                    System.out.println("You chose October");
                    System.out.println("Is this correct? (y/n)");

                    String m1 = stdin.readLine();

                    if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                    {
                        System.out.println("Month =>
October");
                        break;
                    }

                    else
                    {
                        month = 13;
                        continue;

                    }

                }

                else if ( month == 11)
                {
                    System.out.println("You chose November");
                    System.out.println("Is this correct? (y/n)");
                    String m1 = stdin.readLine();

                    if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                    {
                        System.out.println("Month =>
November");
                        break;
```

```
                                        }

                                        else
                                        {
                                                month = 13;
                                                continue;

                                        }

                                }

                                else if ( month == 12)
                                {
                                        System.out.println("You chose December");
                                        System.out.println("Is this correct? (y/n)");

                                        String m1 = stdin.readLine();

                                        if ( m1.equals("y") || m1.equals("Y") ||
m1.equals("yes") || m1.equals("YES"))
                                        {
                                                System.out.println("Month =>
December");
                                                break;
                                        }

                                        else
                                        {
                                                month = 13;
                                                continue;

                                        }

                                }
                        }

                        //Adds the month to the Activity
                        a.setMonth(month);

                        //The user is prompted for the date and a check is provided
to make sure the date is an integer value
                        //if the input is not an integer, the user is reprompted for a
valid input
                        int day = getCheckedInt("Please enter date: ");
```

```
                    //The following three while statements are checks to make
sure the date entered is within the appropriate range for the month entered
                    //If the date does not fall within an appropriate range, the
user is asked to reenter the data
                    while ( (month == 1 || month == 3 || month == 5 || month
== 7 || month == 8 || month == 10 || month == 12) && (day <= 0 || day > 31 ))
                    {
                        System.out.print("This month has 31 days.  Please
enter a number between 0 and 32:");
                        String d2 = stdin.readLine();
                        int day2 = Integer.parseInt(d2);
                        day = day2;
                    }

                    while ( (month == 4 || month == 6 || month == 9 || month
== 11) && (day <= 0 || day > 30 ))
                    {
                        System.out.print("This month has 30 days.  Please
enter a number between 0 and 31:");
                        String d3 = stdin.readLine();
                        int day3 = Integer.parseInt(d3);
                        day = day3;
                    }

                    while ( (month == 2 ) && (day <= 0 || day > 29 ))
                    {
                        System.out.print("This month has 28-29 days.
Please enter a number between 0 and 29:");
                        String d3 = stdin.readLine();
                        int day3 = Integer.parseInt(d3);
                        day = day3;
                    }

                    //The month and date is printed out for the user to see
                    System.out.println();
                    System.out.println("date => " + month + "/" + day);
                    System.out.println();
                    a.setDate(day);

                    //the user is prompted for an hour and there is a check to
make sure the hour is an integer value
                    int hour = getCheckedInt("Please enter hour of start time"+
                                        "(hour-example: If time is 4:45, enter
4): ");
```

```
                    //The while loop checks to see whether or not the hour is
within an aoppropriate range, if not, the user is reprompted to enter a valid hour

                    while ( hour <= 0 || hour > 12)
                    {
                        System.out.println("That is not a valid hour.  The
hour must be between 0 and 13.");
                        System.out.println();
                        System.out.print("Enter a valid hour:");
                        String h2 = stdin.readLine();
                        int hour2 = Integer.parseInt(h2);
                        hour = hour2;
                    }

                    System.out.println();

                    //The user is prompted for minutes and there is a check to
determine whether or not the input was an appropriate integer value
                    int min = getCheckedInt("Please enter minute of start
time"+
                                            "(minutes-example: If time is 4:45,
enter 45): ");

                    while ( min >= 60 || min <=-1)
                    {
                        System.out.println("That is not a valid minute.  The
minutes must be between 0 and 61 minutes.");
                        System.out.println();
                        System.out.println("Enter a valid minute:");
                        String m2 = stdin.readLine();
                        int min2 = Integer.parseInt(m2);
                        min = min2;
                    }

                    //While printing, if the user selected a value less than ten
for minutes, a zero is added in front of it so it is easier to read
                    if (min <= 9)
                    {
                        System.out.println("start time => " + hour+ ":"+
"0"+ min);
                    }

                    else
                    {
                        System.out.println("start time => " + hour+ ":" +
min);
```

```
                    }

                    //The start time hour and minutes are set
                    a.setStartTimeh(hour);
                    a.setStartTimem(min);

                    //The user is prompted for a location
                    System.out.print("Enter location: ");
                    String location = stdin.readLine();
                    a.setLocation(location);

                    //The user is prompted for notes
                    System.out.print("Enter notes: ");
                    String notes = stdin.readLine();
                    a.setNotes(notes);
                    //The Activity and its attributes are added to the Linkedlist
                    Cal.add(a);

                    //The user is prompted for a file name and it is saved
                    System.out.println("Please enter a filename:");
                    Cal.save(filename);
                }

                //If the user selects the print option, the data is sorted by month and
date, then printed
                else if (answer.equals("p"))
                {
                    System.out.println("User selected PRINT");
                    Cal.selectionSort();
                    Cal.print();
                    System.out.println();

                }

                //If the user selects the find option, they are provided with a new
menu and asked to select a search method
                else if (answer.equals("f"))
                {
                    System.out.println("User selected FIND");
                    System.out.println();
                    System.out.println("What would you like to search by?");
                    System.out.println("a) activity name");
                    System.out.println("b) activity location");
                    System.out.println("c) activity date");
                    String search = stdin.readLine();
```

```
                        if (search.equals("a"))
                        {
                                System.out.println("Search by Activity Name");
                                System.out.println("Enter name:");
                                String searchName = stdin.readLine();
                                System.out.print(Cal.findName(searchName, node,
0));
                        }

                        if (search.equals("b"))
                        {
                                System.out.println("Search by Activity Location");
                                System.out.println("Enter location:");
                                String searchLocation = stdin.readLine();

        System.out.print(Cal.findLocation(searchLocation));
                        }

                        if (search.equals("c"))
                        {
                                System.out.println("Search by Activity Date");
                                int searchMonth = getCheckedInt("Please enter
month (example: If month is December, enter 12): ");
                                int searchDate = getCheckedInt("Please date:");
                                System.out.print(Cal.findDate(searchMonth,
searchDate));
                        }
                }

                //The user selects the remove option, the user must enter the name
of the activity to be removed
                else if (answer.equalts ("r"))
                {
                        System.out.println("User selected remove");
                        System.out.println("Enter Activity Name:");
                        String removeName = stdin.readLine();
                        Cal.remove(removeName);

                }

                //If the user selects the quit option, the program is exited
                else if (answer.equals("q"))
                {
                        System.out.println("User selected QUIT");
                        break;
                }
```

```
                else
                {
                        continue;
                }

        }
    }

    //This method checks whether or not the data inputed is an integer value, if it is not, the user is reprompted to enter a valid integer
    public static int getCheckedInt(String prompt) throws IOException
    {
        String data = "";
        int x = 0;

        for(;;)
        {
            try
            {
                System.out.print(prompt);
                data = stdin.readLine();
                x = Integer.parseInt(data);
                break;
            }
            catch (NumberFormatException error)
            {
                System.out.println("ERROR: Please enter valid number:");
            }
        }
        return x;
    }
}
```

C2: Usability

Feature	Documentation
Quick Searches	See screenshots on page 51, 53
Easy to follow interfaces	See screenshot on page 49-53
Input Error Detection	See screenshot on page 49-50
User-Friendly Menus	See screenshot on page 48, 51
Store and Display information in chronological order	See screenshot on page 49

C3: Handling Errors

Error	Location
Removing From an Empty List	A check is placed in the remove functions to look to see whether or not the calendar is empty before removing from the list. See code on page 24
Data Type Error	If the user inputs a data type that does not comply with what is needed (for example, if the user tries to input a string in the place of an integer) the user is prompted to reenter the data. See screen shot on page 49
Date Entry	If users enter months or dates in the form of a String they are asked to reenter data. Also, if the date entered does not fall within the range of days within that month, generally 1-31, they are asked re-enter data. See screen shot on page 49-50
Invalid choices from the menu	If users choose options which are not listed in the menu, the menu is printed out again and the user is once more prompted to select an option from the menu. See screen shot on page 52
Time Entry	If users enter an hour or minutes that do not fall within a reasonable range (1-12 hours and 1-59 minutes), then users are prompted to reenter a valid time. See screen shot on page 50
Month Entry	Users are asked to verify the month entered to make sure it is the month they desired to enter. See screen shot on page 49
Visibility	The user actions are printed (when dates and times are added) so the user can keep track of what he/she is doing. See screen shot on page48

C4: Success of Program

Objective	Evidence
Collects information from user (asks the user to input attributes of the activity)	See screen shot on page 48
Visually appealing output	See screen shot on page 48
Invalid choices from the menu (the user prompted for another choice if they make a mistake entering an option)	See screen shot on page 51
Quick Searches (The user is able to conduct quick searches to find activities based on name, date, and location)	See screen shot on page 50-51
Displays data in order (The user is able to select "print" option to print data in order of date.)	See screen shot on page 49
User can add/remove/edit information (The user can select "add" "remove" or "change" to manipulate the database.)	See screen shot on page 48, 52

D1: Test Output (Annotated Hard Copy)

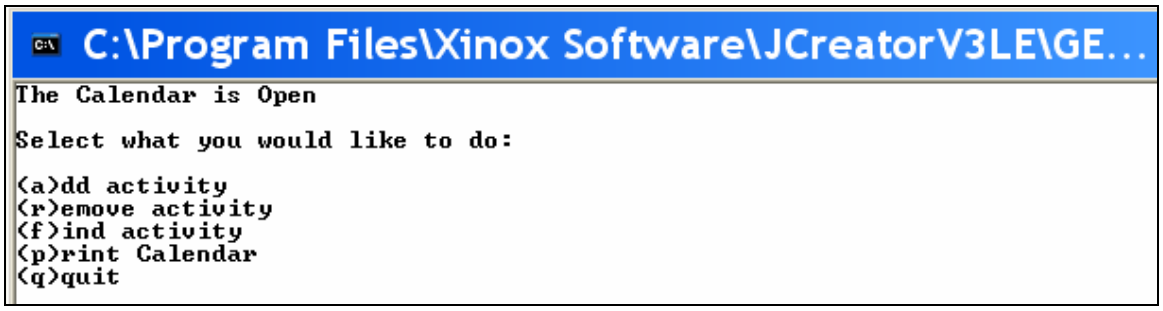


Figure 1: The main menu provides the user with options. It is clear and easy to read.

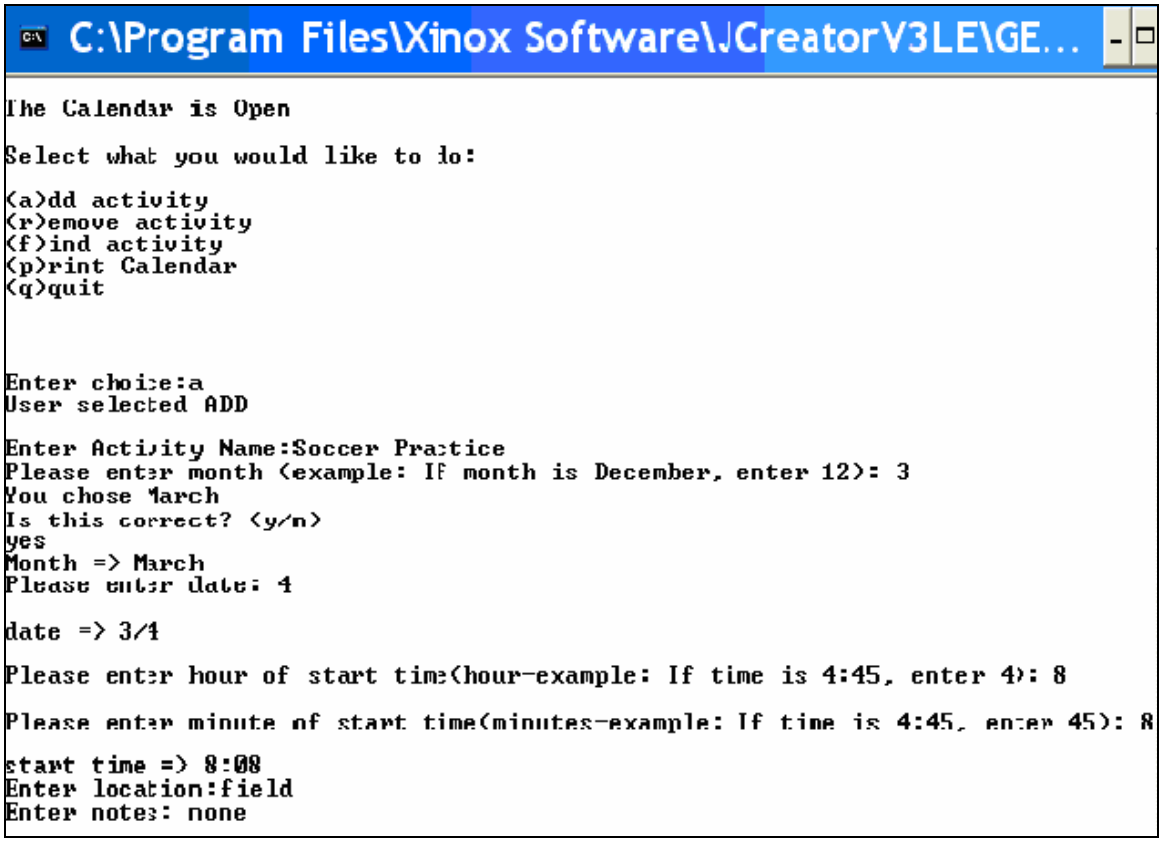


Figure 2: This shows the addition of an activity to the database. The activity name (soccer practice), date (March 4), time (8:08), location (field), and notes (none) were entered into the list. The user was prompted for each attribute and provided with helpful instructions, including examples of inputs.

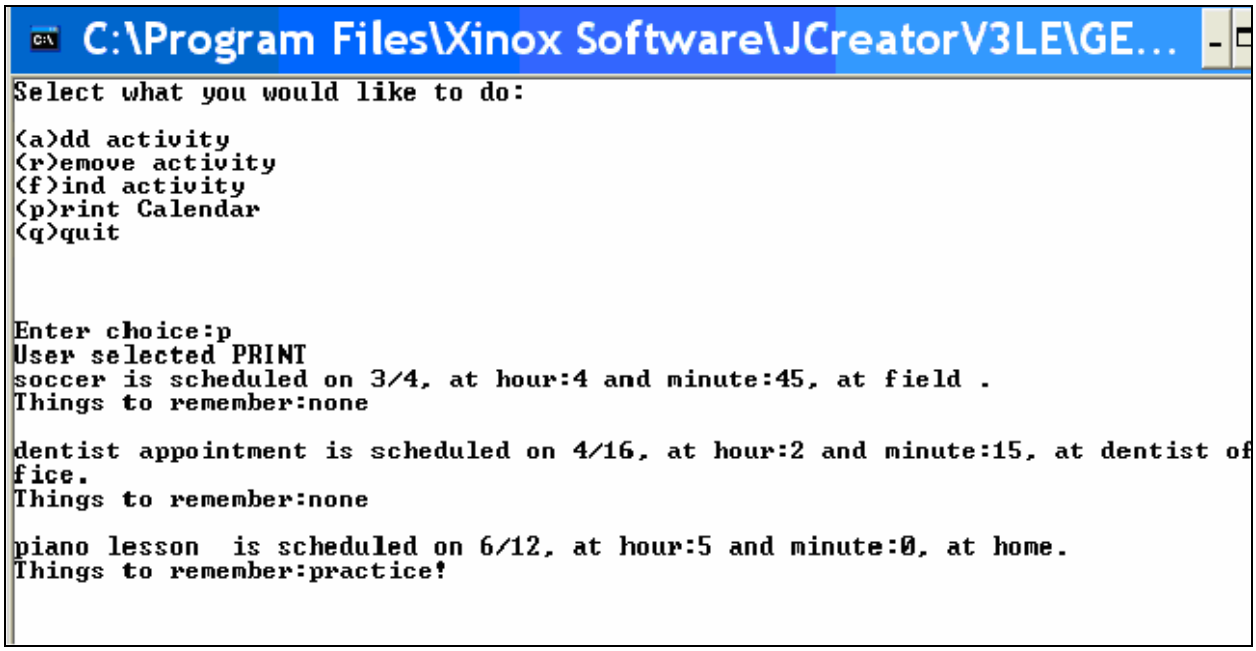


Figure 3: Data that had been previously entered has been printed in chronological order after the "print" option was chosen from the menu.

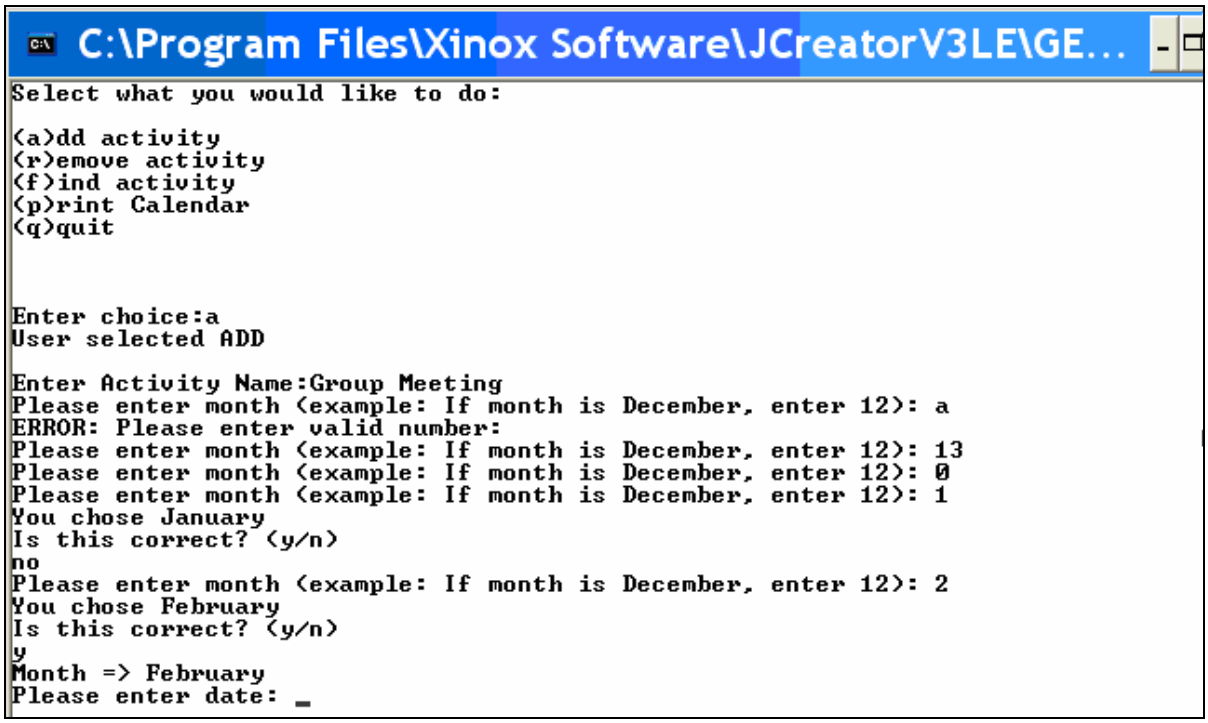


Figure 4: An activity (group meeting) is being added to the database. When the user inputs the month, the input type is checked. The input has to be an integer, and if it is not (i.e. if it is a String like "a"), then the user is asked to enter a valid number. The range is also checked, if the user inputs a month less than one or greater than twelve, the user is again asked to enter a valid number. Once a valid input has been made, it is verified by a prompt asking whether or not the month is correct. Finally, the final month choice is printed out.

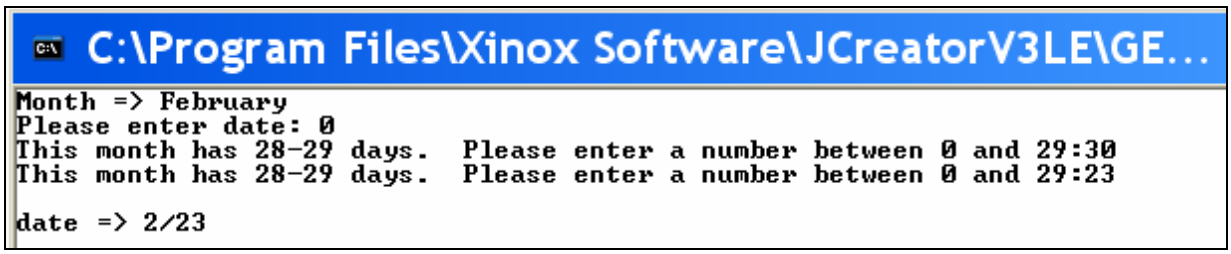


Figure 5: The date is being entered for an activity. The date entered must fall within the appropriate range if the user inputs a date less than or greater than the minimum and maximum acceptable dates for the month, the user is asked enter a valid date within the appropriate range. The final date selection is printed out.

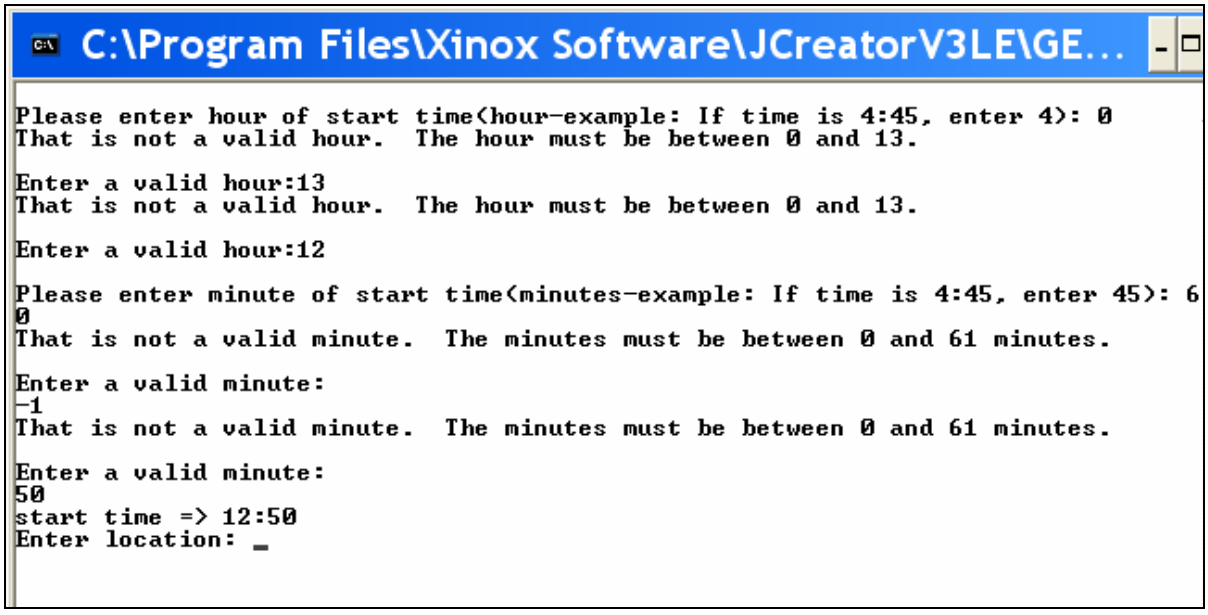


Figure 6: The time is being entered for an activity. The hour of the start time must be within a reasonable range (1-12) and the minutes must also be within a reasonable range (0-60); if the user inputs times outside of the range, they are prompted to input valid data. The final start time is printed.

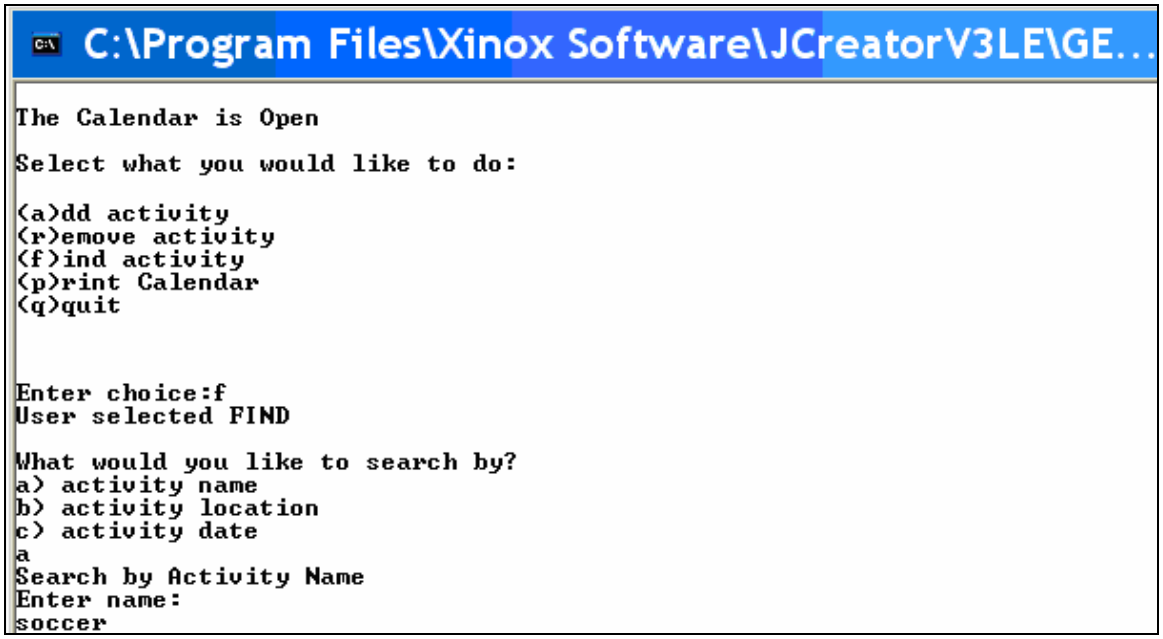


Figure 7: If the user chooses to search for an activity, by selecting the "find" option from the menu, they user may search by the activity's name, location, or date.

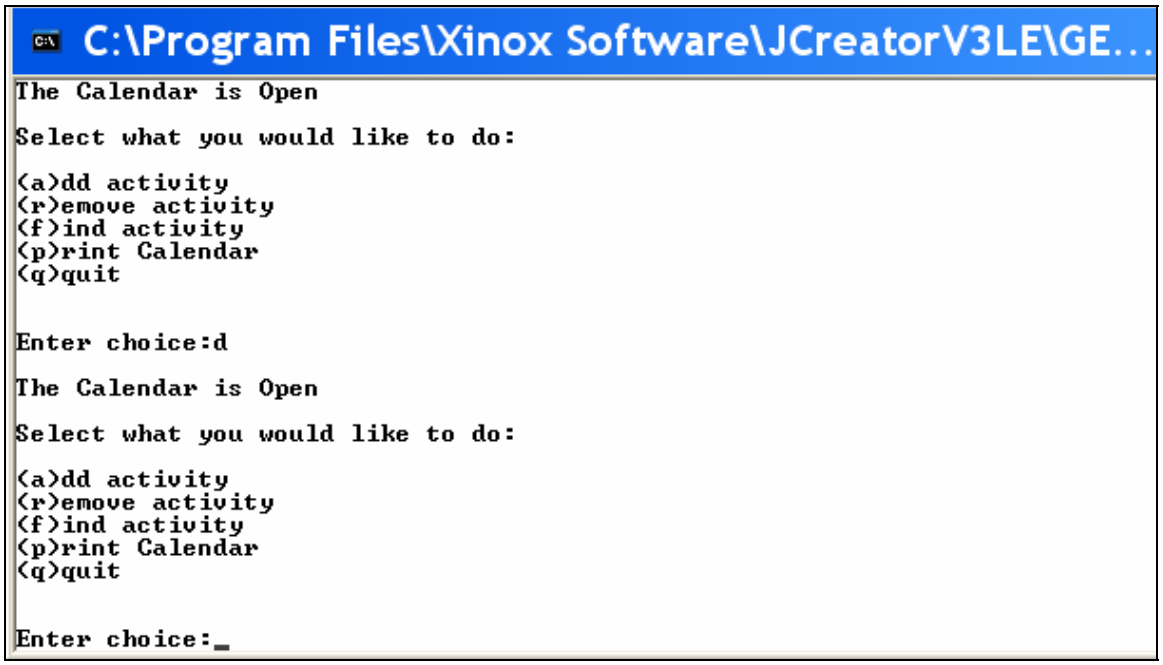


Figure 8: If the user selects an option that is not in the menu, then the calendar is printed out again and the user is again prompted to select a choice.

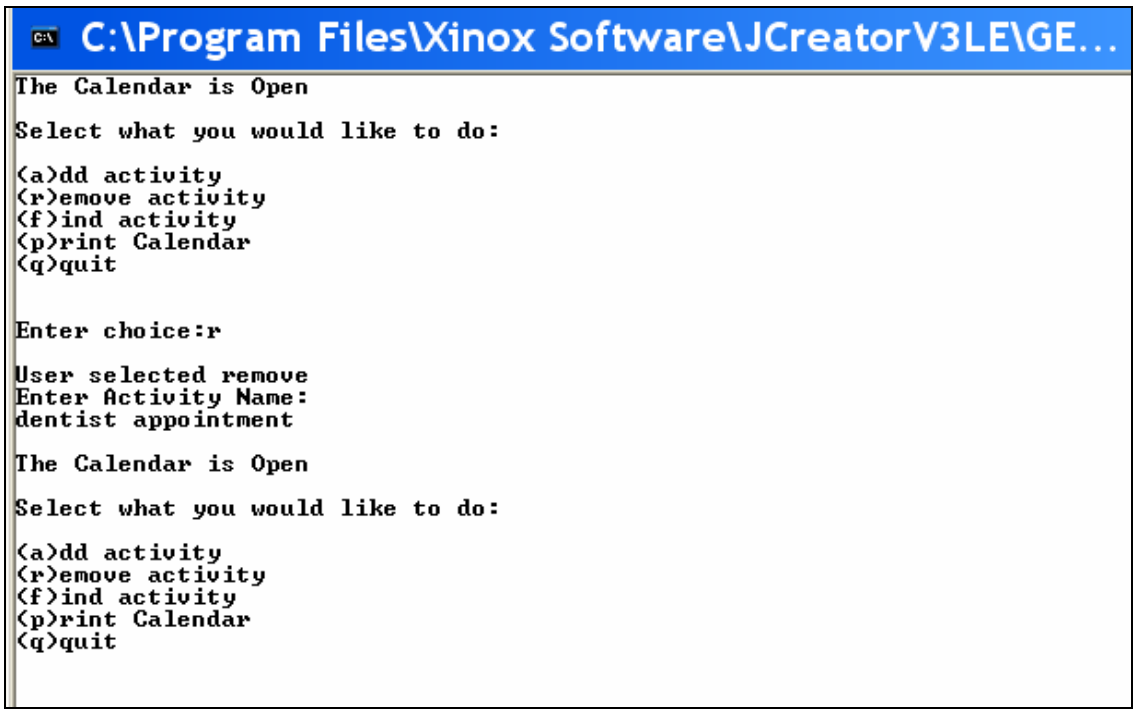


Figure 9: If the user selects to remove an activity, the user is prompted for the activity name.

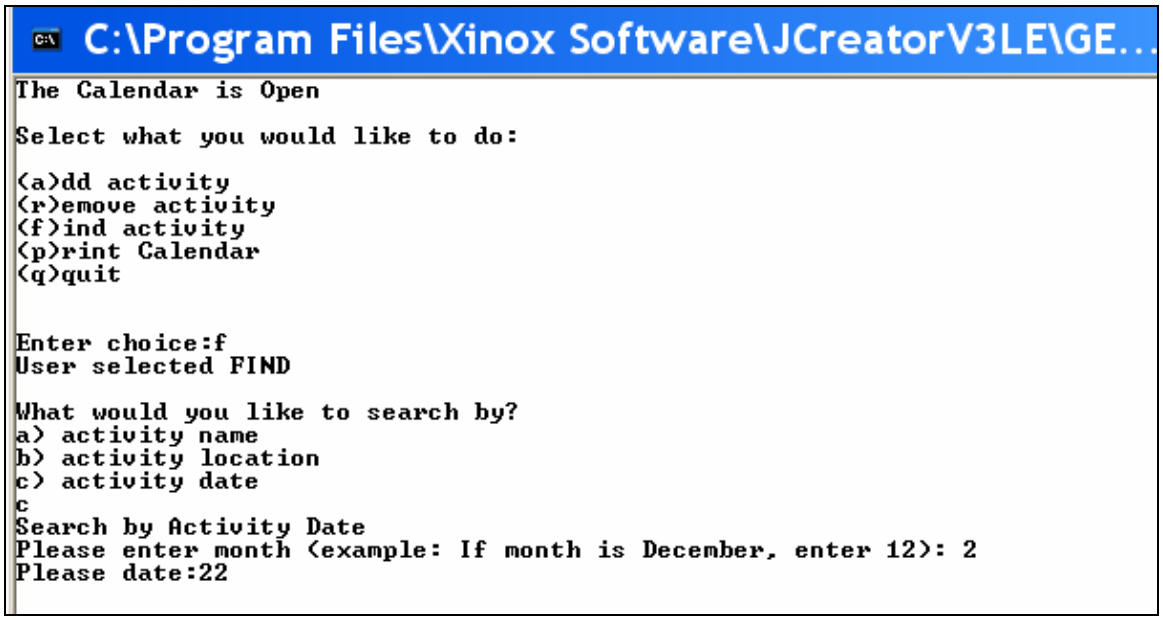


Figure 10: If the user selects to find an activity or search for an activity by date, the user is prompted for the month and date of the activity.

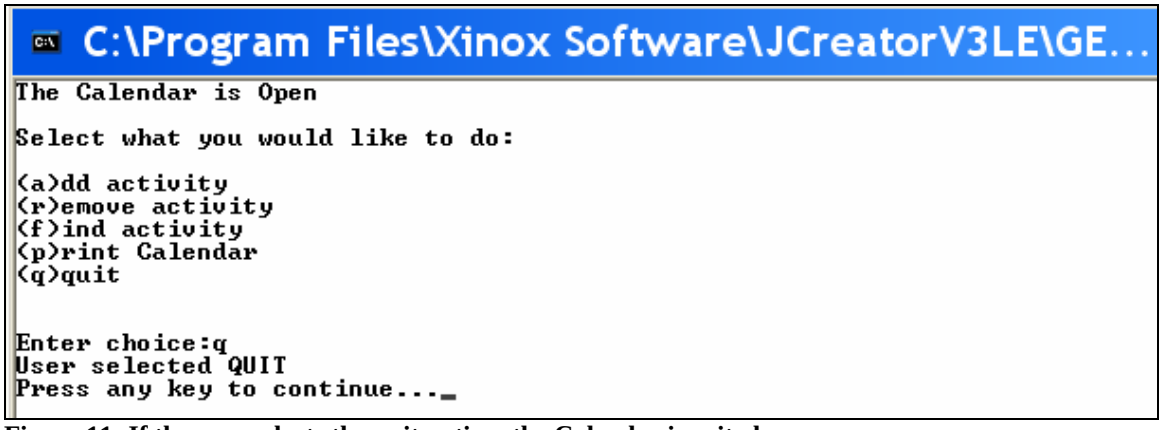


Figure 11: If the user selects the quit option, the Calendar is exited.

D2: Evaluation of Solution

The program addressed the criteria for success in having efficient searching, easy to follow interfaces, input error detection, and saving and loading files. The efficient searches were due to the use of a linked list to store data, rather than static arrays. The interfaces included many simple menus and prompts for input. The input was checked for error by having checks to determine whether or not the input type was correct and by printing the inputted data and asking the user whether the data was correct. There were many opportunities for users to correct data.

The initial design was appropriate in addressing the criteria of success and in creating this program. It allowed for activities to be compared, updated, added, removed, and found based on different attributes. It also allowed for the database to be printed. O

Despite this, there were still limitations to the program. For example, the program only stored 12 months in a year and did not allow users to input data for more than one year. If two activities of the same name existed in the database, and the user wanted to remove one, only the first activity would be deleted. The user did not have visability in all stages. For example, the user could not immediately see what he/she had added to the database without printing out the entire database to check. There was no way for a user to edit an activity, just a way for the user to add/remove. The user also did not have the ability to compare activity times to determine where overlaps occur.

These limitations can be addressed and the program can be enhanced through the addition of different components. For example, the addition of different user accounts and passwords would allow for more than one person to use the program. Allowing the program to include different years will expand the length of time the program can be used for. A comparison of end times as well as start times will make the program more useful in preventing scheduling conflicts. Having the option of “AM” or “PM” time will help the user be able to better keep track of schedules, rather than just having to choose from 1-12. A calculation of the amount of time available to the user each day, after an activity has been added to that day, would also help with scheduling. A graphic user-interface may make the system more appealing to users and provide images of a calendar that users are more accustomed to.

Also, to improve the user's visual understanding of the inputted data, it can be printed after files are loaded. The overall visibility of the Calendar Database may be improved so the user can tell if things are working (adding, finding, removing, etc). It may help users if more attributes are included for an activity, such as the people involved. It would also be useful for users to be able to enter the attribute “frequency” of each activity to avoid entering the same information multiple times or to have a system of “priority numbers” to help understand the importance of each event when solving scheduling conflicts. If there are more than one matches for an activity the user is trying to remove, the user should be provided with an option of which to remove.

Another suggestion in creating the program may be to have separate code of the interface and code the data. This way the code for the data does not have to be changed if code for the interface needs to be changed. There should also be a way to save data as the user enters it so that if the program crashes or the user accidently exits the program, they can retrieve what they had done so far.

The program's methods had Big O complexities of O(log2n) for the find, sort, and delete methods. The print, compare, add and save methods had Big O complexities of O(n). The load method had a Big O complexity of 0(1).

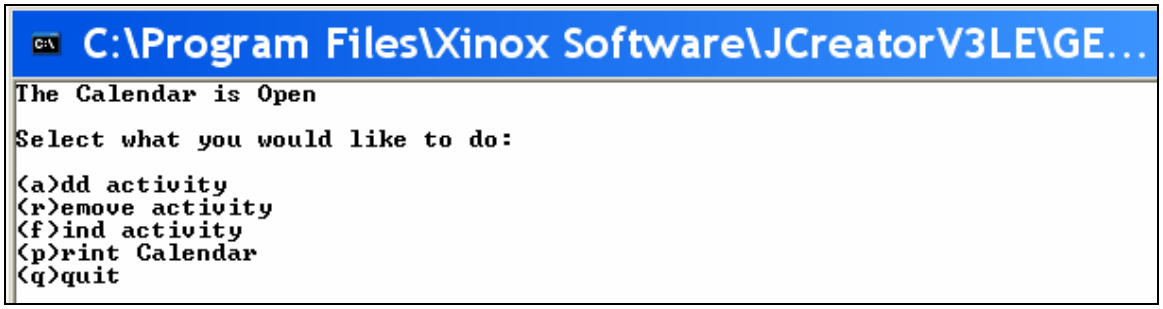
D3: User Documentation (User Manual)

Installation

1. Install Java
2. Download and save all files in Calendar Database (ActivityNode.java; Activity.java; CalendarL.java; CalendarMain.java; ActivityLL.java).
3. In the terminal, type javac and the file name. For example, ActivityNode would be: javac ActivityNode.java. After each javac, press the Enter key.
4. Once each file has been entered into the terminal with javac, type in java and the file name. For example, ActivityNode would be typed in: java ActivityNode. After typing in each file name, press the Enter key.
5. To begin using the database, type in "java CalendarMain"

Add Activity

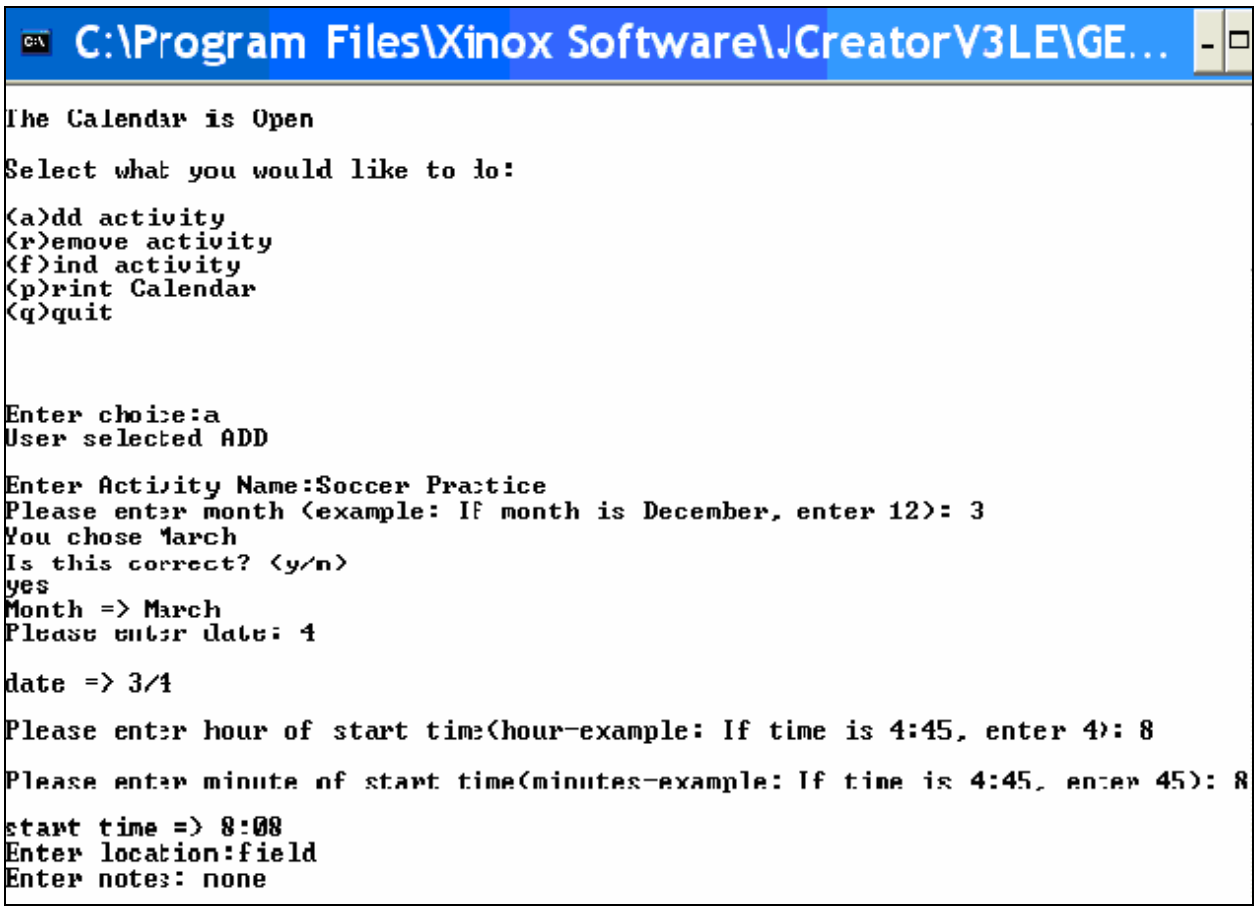
1. Once "java CalendarMain" has been typed into the terminal, a menu will appear:



2. Type in "a" and press the Enter key.
3. You will then be prompted for an activity name. Type in what you would like the activity to be called in the calendar database, then press the Enter key. (In the example below "Practice" is the activity name).
4. Next, you will be prompted for a month. Type in the numerical representation of the month you would like to put your activity in and press the Enter key. In the example the user chose March, so he/she entered "3."
5. Once the month is entered you will be prompted to verify the month. Type in "yes" or "y" if the month printed on the screen is correct and press Enter. Type in "no" or "n" if the month is in corrected , press Enter, then go back to step 4.
6. Then, you will be prompted for a date. Type in the number of the date you would like to put your activity in and press the Enter key. In the example, the date entered is "4."
7. Next, you will be prompted for the hour your activity is to start. The hour has to be 1 to 12 hours. In the example below, the hour is "8." Please enter a number for the hour and press the Enter key.
8. Then you will be prompted for the minute your activity will start. This is a number between 0 and 59. In the example below, the activity starts at 8 and 0 minutes, or

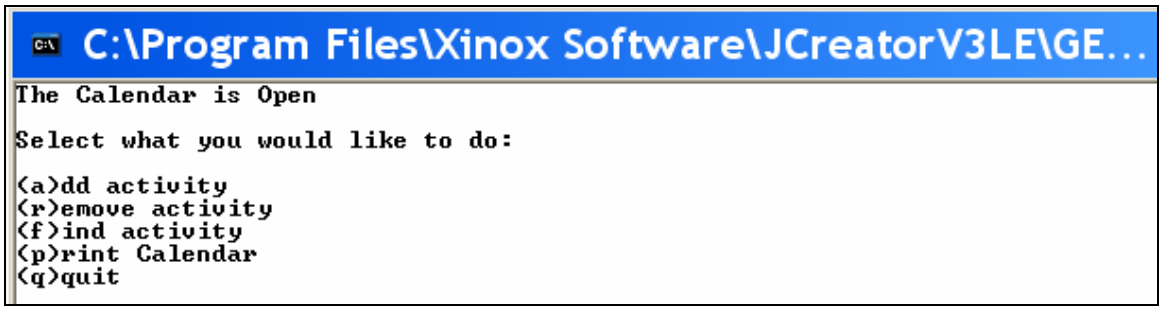
8:00, so the user entered "0" for minutes. Be sure to press the Enter key once you have typed in your number.

9. When the time you have entered is shown on the screen (example: "=> 8:00" in the sample below), you will be asked to enter the location. Type in where the activity will occur and then press the Enter key. The activity will occur at "field" in the activity below.
10. Then, you will be asked to type in any notes. There is no limit to what you may type in here, but it is best to keep your notes succinct. If you have no notes to add, you may type "none" as in the example below. Then press the Enter key.
11. You have successfully added an activity! Once the activity has been added, you will be shown the menu, as shown above. To check to see if your addition has worked, you may select the print option (Go to Print Calendar for more details on how to do this).

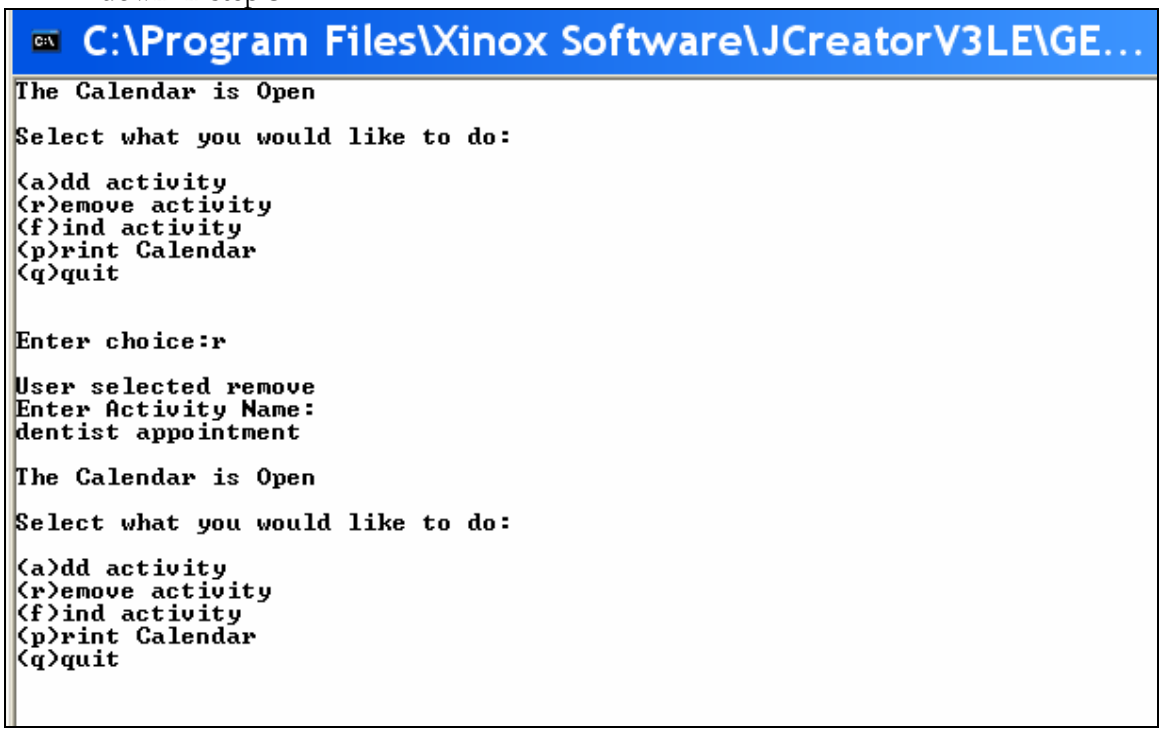


Remove Activity

1. Type in "java CalendarMain" into the terminal. Once you have typed this and pressed the Enter key, a menu will appear (shown below).

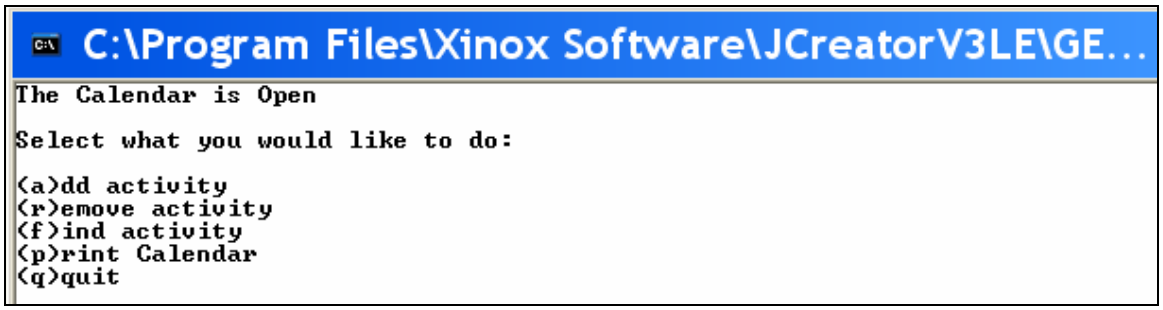


2. Type in "r" and press the Enter key.
3. You will then be prompted for an activity name. Type in the name of the activity you would like to remove and press Enter. In the example below, the user chose to remove "dentist appointment."
4. Once the activity has been removed, you will see the menu again ("Calendar is Open…"). To check that the remove has been successful, you may select the print option (Go to Print Calendar for more details on how to do this).
5. If you find that the remove has not been successful due to a spelling error or whatever reason, closely check the activities that have been printed, write down the name of the activity you wish to delete, and go to step 2, using the name you have written down in step 3.

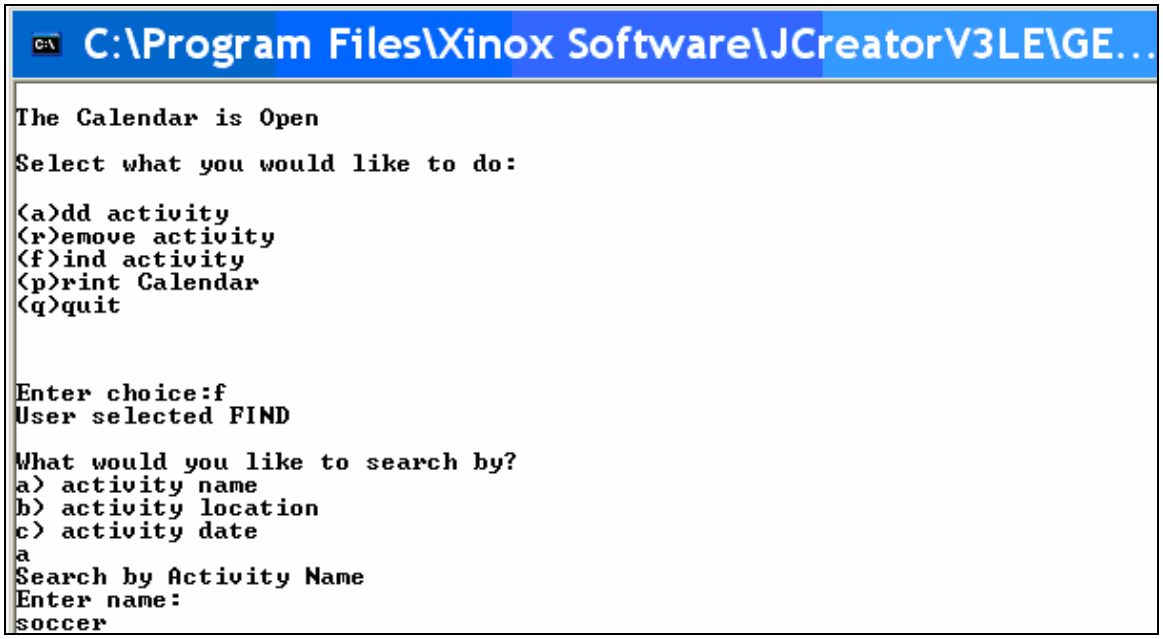


Find Activity

1. Type in "java CalendarMain" into the terminal. Once you have typed this and pressed the Enter key, a menu will appear (shown below).



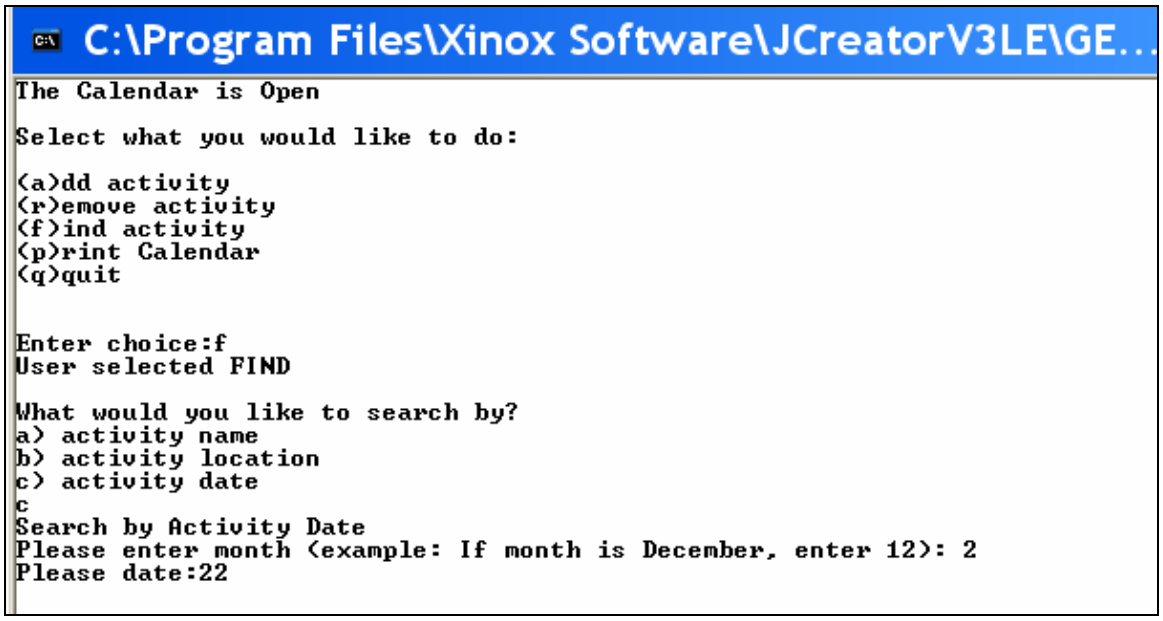
2. Type in "f" and press the Enter key.
3. A new menu you appear asking you what you would like to search by. You can search for an activity by the name of the activity, the location it will occur as recorded in the Calendar database, or the date (month and day) it will occur as recorded in the database. Type in "a" if you would like to search by the name. Type in "b" if you would like to search by location. Type in "c" if you would like to search by date. Then press the Enter key. In the example below the user has selected to search by activity name.



4. You will then be asked to enter the attribute you are searching by. For example, if you have chosen to search by the activity name, you will be asked to enter the activity

name you are looking for, then you will press the Enter key. In the example above, the user typed in "soccer" when asked to enter the attribute.

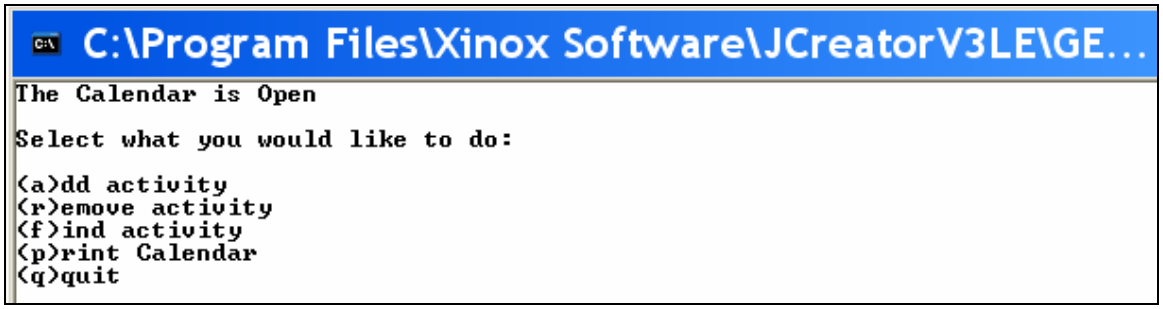
5. The results of your search will be printed out. If your search did not provide satisfactory results, go back to step 2 and try again or go print the calendar (go to Print Calendar for more details) to print the whole calendar and look for your activity in the printed list.



The above picture demonstrates the user entering data for an activity search by date.

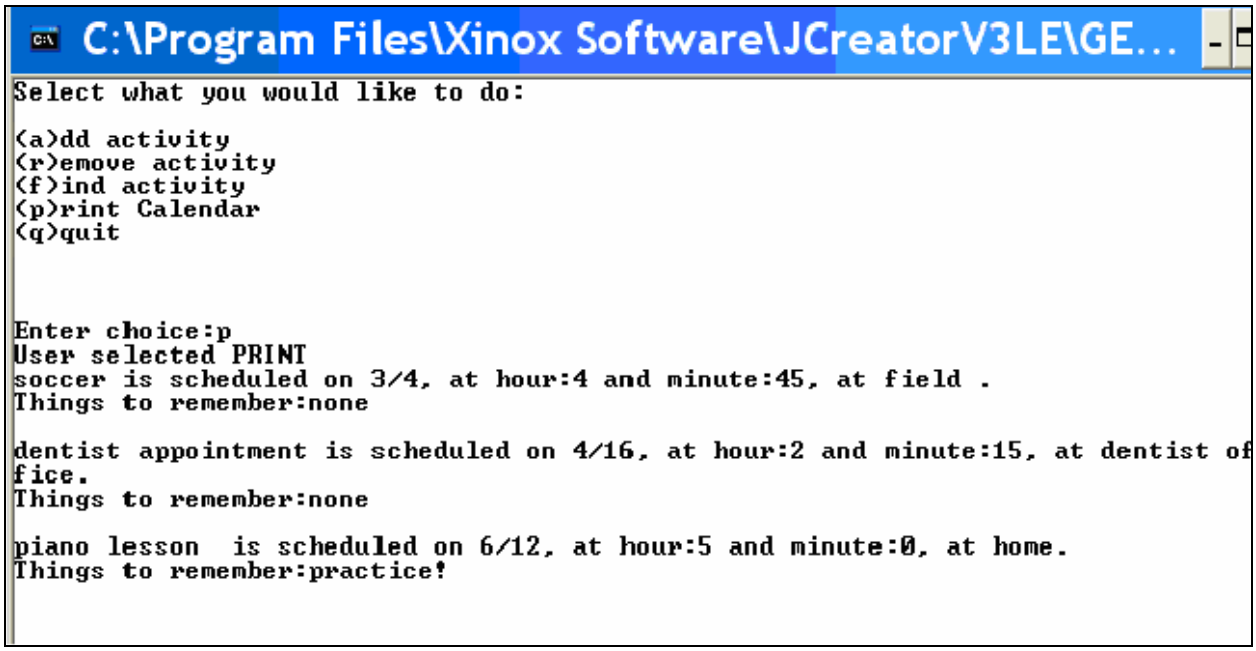
Print Calendar

1. Type in "java CalendarMain" into the terminal. Once you have typed this and pressed the Enter key, a menu will appear (shown below).



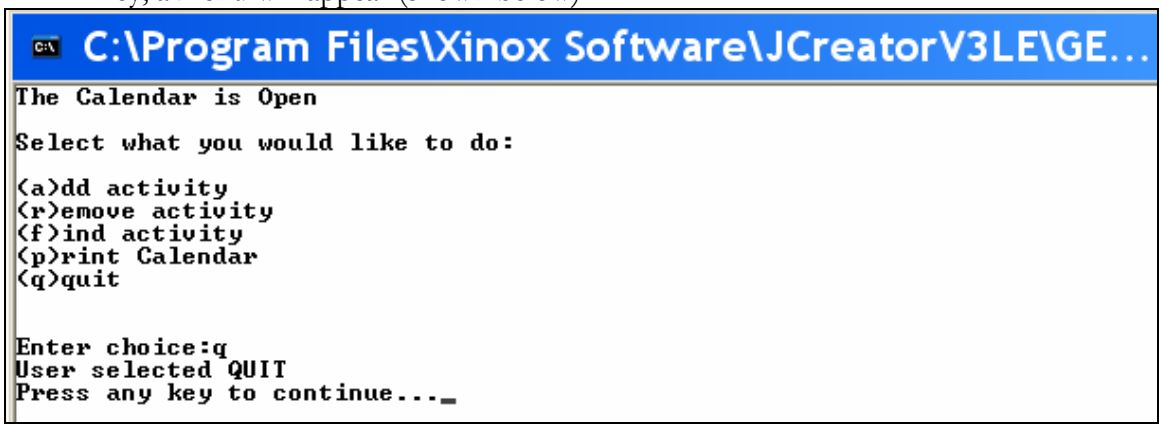
2. Type in "p" and press the Enter key.
3. The calendar will be printed out in chronological order by month and date (example shown below).

4. If nothing appears besides the menu again, this may mean you have not added any activities to print. Go back to Add Activity section, follow steps, and then come back to Print Calendar.



Quit Calendar

1. To quit the calendar, the program must be running already. If it is not, type in "java CalendarMain" into the terminal. Once you have typed this and pressed the Enter key, a menu will appear (shown below).



2. From the menu, select the quit option by typing in "q" and pressing the Enter key.
3. You will either be immediately exited from the program or you will see "Press any key to continue…" as in the above example. To exit after the "Press any key to continue…" message has been displayed, press any key on your keyboard.

Mastery Aspects

Mastery Aspect	Where	How	Why
Recursion	Page 26	Recursion is used to find an activity by name in the findName method by having the function call itself in the find method over and over until the Activity is found.	The recursive function allows the find method to occur as many times as needed for the Activity to be found.
Polymorphism	Page 29-44	Polymorphism is used by calling functions from one class in another.	This allows the code to be organized and clear.
Encapsulation	Page 21	Encapsulation is used in the Activity class, in which each activity has several attributes.	This allows many attributes to be assigned to each activity, which creates organization in storing necessary data about each activity.
Parsing a text file or other data stream	Page 32-45	Strings are parsed to integers when data is inputted and checks are in place to determine whether or not the input is appropriate.	Parsing strings to integers allows data to be manipulated and input to be converted for the users. Converting to integers allows calculations to be made later using the input.
The use of any five standard level mastery factors	Page 21-44	• User defined Objects: There are defined objects (i.e. activity) • Simple Selection: There are many loops, especially in the menu that use if-else to determine what the user has selected and what further actions need to be taken. • User-defined methods with parameters: There are methods in the linked list which take in parameters. • File Input/Output:	• Having user defined objects allows for more freedom and organization in the code. • The if loops allow for an easy way to determine what choice the user selected. • Having user defined methods with parameters allow the user's input to be used in the methods. • File I/O allows data to be saved. • Sorting allows for the data to be organized and adds

		File I/O is used to read and write data. • Sorting: Sorting is used in the linked list to sort through the activities and put them in chronological order by date. • Searching: Searching is used in the linked list to find specific activities.	aesthetic appeal to the output for the users. • Searching allows for users to pull out different data that was previously entered.
Implementing a hierarchical composite data structure	Page 21-44	A hierarchical composite data structure is used throughout the code because there is a hierarchy in the classes (calendar which stores activity linked list which stores node which stores activity which stores the activities' attributes)	A hierarchical composite data structure allows for organization and different elements to be contained and accessed in different classes.
Abstract data types	Page 24-32	A Linked List was used to store data.	The linked list allows for fast addition of data to a list and allows for easy manipulation of data within the list.