GROUP 4

CYCLIC DOUBLY LINKED LIST

Alyssa Hardy | Bryce Hankinson | Grant Solomon | Samuel Davis

THE PROGRAM

FEATURES OF SONIC MEDIA PLAYER

- Basic function buttons: play, pause, stop, forward, and backwards
- A table showing all the songs loaded into the media player
- Shuffle, repeat all, and repeat one features
- Sorting songs by title, artist, album, genre, and length
- A progress bar which shows the the song’s progress. Clicking on it will allow you to skip to any location in the song.
- Labels showing previous, current, and next songs to emphasize the Cyclic Double Link List’s nature

THE CODE

LIST IMPLEMENTATION

- The global songList variable is created like a normal ArrayList along with three global nodes variables to allow the program to keep track of the previous, current, and next songs

```
Node<Song> currentSong, previousSong, nextSong;
CyclicDoublyLinkedList<Song> songList = new CyclicDoublyLinkedList<>();
```

- When the program starts, songs are pulled from their directory and put into the list using the addToEnd() method

Loading songs into the list

```
//Create song Files
File f = new File("src/ext/songs");
File[] musicFile = f.listFiles();

//Add songs into CDLL
for (int i = 0; i < musicFile.length; i++){
    Song s = new Song(musicFile[i].getName(), musicFile[i].getPath());
    songList.addToEnd(s);
}
```

Assigning the nodes

```
currentSong = songList.getHead();
previousSong = currentSong.getPrevious();
nextSong = currentSong.getNext();
```

```
public void addToEnd(E e){
    Node<E> node = new Node<>(e);
    if(tail == null){
        head = node;
        tail = node;
        node.setNext(node);
        node.setPrevious(node);
    }
    else{
        node.setPrevious(tail);
        tail.setNext(node);
        head.setPrevious(node);
        node.setNext(head);
        tail = node;
    }
    size++;
}
```

SORTING THE LIST

- We chose to use bubble sorting for our because it's a relatively simple sorting method that could be duplicated easily with each parameter
- It has an average-case complexity of O (nlogn)
- Each node in the list is compared to the next, and if out of order, the two are swapped. Iteration continues through the list until it's sorted

```
public void sortListByTitle() {
    if (!(getElement() instanceof Song)) return;
    Object temp = null;
    //Check whether list is empty
    if(head != null) {
        Node current = head, index = head;
        for(int i = 0; i < size; i++){
            Song currentSong = (Song)current.getElement();
            for(int j = 0; j < size; j++){
                Song indexSong = (Song)index.getElement();

                if (currentSong.comparebyTitle(indexSong) > 0) {
                    temp = current.getElement();
                    current.setElement(index.getElement());
                    index.setElement(temp);
                }
                index = index.getNext();
            }
            current = current.getNext();
        }
    }
}
```

- On the program side, the sortSongTitle() method calls the list's sortListByTitle() method, and then updates the table to show the sorted songs

```
private void sortSongTitle(){
    songList.sortListByTitle();

    for (int i = 0; i < songList.size(); i++){
        Song s = (Song)songList.getElementByIndex(i).getElement();

        String time = String.format("%d:%02d",s.getLength()/60,s.getLength()%60);
        tableModel.addRow(new Object[]{s.getTitle(),s.getArtist(),time,s.getAlbum(),s.getGenre(),s});
    }
}
```

- This is repeated to allow sorting via album, artist, genre, and length as well

CLASS MODELS

CLASS MODELS

AudioPlayer
-currentFrame : Long -clip : Clip -ais : AudioInputStream -song : Song -status : String
+audioPlayer(song : Song) +getSong() : Song +getStatus() : String +pause() +play() +restart() +resume() +update() -resetAudioStream()

Song
-fileName : String -filePath : String -file : File -title : String -artist : String -album : String -genre : String -lengthSeconds : int
+Song(fileName : String, filePath : String) +compareByAlbum(other : Song) +compareByArtist(other : Song) +compareByGenre(other : Song) +compareByLength(other : Song) +compareByTitle(other : Song) +equals(o : Object) : boolean -parseSong(fileName : String)

CLASS MODELS

GUI
-currentSong : Node<Song> -nextSong : Node<Song> -previousSong : Node<Song> -songList : CyclicDoublyLinkedList<Song>
-createSongList() -createSongTable() -init(song : Song) -forward() -back() -sortSongAlbum() -sortSongArtist() -sortSongGenre() -sortSongLength() -sortSongTitle() -shuffle() -updateLabels()

ADDITIONAL NOTES

WHY A CYCLIC DOUBLY LINKED LIST WORKS WELL FOR A MEDIA PLAYER

- Because of the circular nature of the list, the media player can cycle through songs via the repeat all feature until stopped by the user
- Being able to traverse the list both forwards and backwards is ideal for media player functionality
- Each node is directly connected to both the previous and next nodes, which allows quick and easy access to fast forward and backwards without delay

FEATURES TO ADD

- A search feature by title, artist, album, and genre
- The ability to add and remove songs from the song library
- Create and edit playlists
- A “play next” button that allows the user to create a song queue

CONCLUSION

TO RECAP...

- A Cyclic Doubly Linked List is ideal for a media player because:
  - It has dynamic sizing
  - Its nodes are easily accessible, allowing for repeated traversal from any point in the list going either direction
  - Adding nodes is quick and efficient with a constant time of O(1)