Gruppuppgift DA353A VT17

Grupp 4

Medlemmar
Tom Leonardsson
Carl Zetterberg
David Svensson
Rada Alasadi

Innehållsförteckning

Arbetsbeskrivning

Tom Leonardsson
”jag har jobbat med att skapa somliga datastrukturer så som List och Hashtable, andra har återvinns från de vi har jobbat med på labbar. Har även jobbat minimalt med GUI för att få paneler att gömmas och vissas när man loggar in, att ha alla i en skärm och att det skrivs ut vad man har lånat. Gjorde också så att man kunde logga-in, låna, lämna tillbaka och söka bland media för att låna. Controller-view stilen blev rätt så klöddig detso längre in man kom för det blev svårt att se till att alla controllers kom åt korrekt data och andra kompontenter. ”
-Tom Leonardsson

Carl Zetterberg
” Jag har under detta grupparbete skapat klasserna dvd och book som är såkallade media objekt , båda är media objekt men har olika attribut eftersom att en bok t.ex. har en författare och en dvd/film har skådespelare. Har även jobbat med denna rapport och färdig ställt den och med hjälp av vad gruppen summerat om sig själva och givit mig sina sekvensdiagram.
- Carl Zetterberg

David Svensson
”Under uppgiftens gång har jag arbetat mycket med User, detta behandlar hur en användarens uppgifter lagras, såsom telefonnummer, namn och Id samt vilket objekt användaren lånat och placera detta i listan för lånade objekt. Jag har fått kommunicera mycket med andra i gruppen för att få detta att fungera korrekt. När något varit komplicerat har jag vänt mig till Tom som är mycket duktig på java programmering” – David Svensson

Rada Alasadi
”Jag arbetade med att skriva gui:en till hela programmet. Jag började första med att rita en bild på hur de olika fönsterna skulle se ut. Jag delade upp alla funktioner in varsitt fönster, det gör det mer synligt för användaren vad han/hon gör i beroende vad användaren skall göra. Tom, en av mina gruppkamrater, kom på iden att lägga alla fönster i ett enda stort fönster, och sedan använda sig av flickar för att byta mellan lån- och återlämna fönster. Iden var mycket bra då det blir mycket mindre krångligt för användaren och användarvänligt. Efter att jag va klar med att skriva koden för fönstren så började jag koppla ihop dem med respektive controller. ” - Rada Alasadi

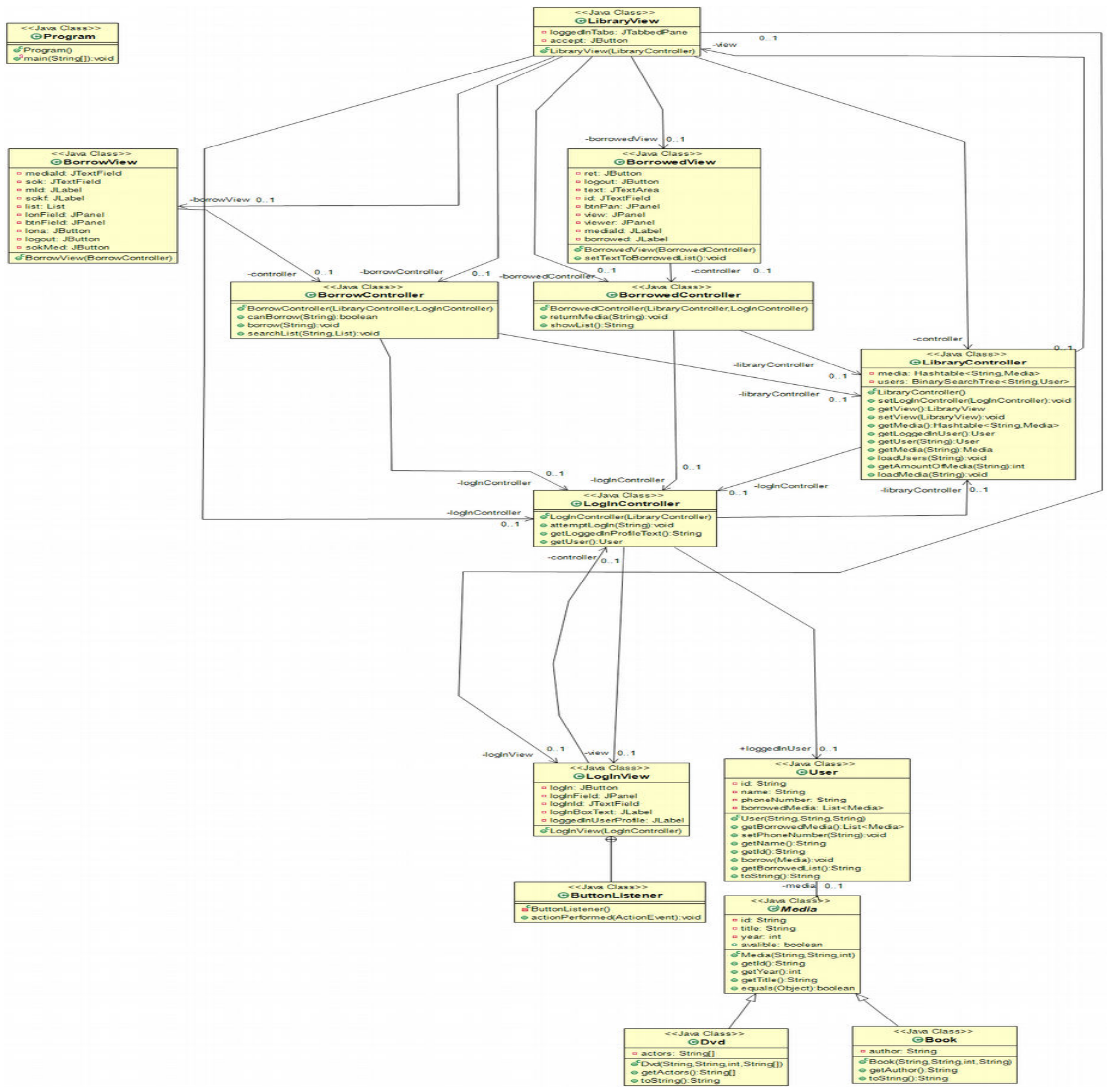
Instruktioner för programstart

För att kunna starta programmet så måste alla medföljande filer finnas i sina mappar (collections och library). Den klassen som startar programmet sedan är Program.java som finns i library mappen.

Systembeskrivning

Vi har som grupp skapat ett program som fungerar som ett virtuellt bibliotek med syftet att kunna logga in som en användare och sedan söka sig fram till en bok eller dvd som man vill låna och sen kunna lämna tillbaks den. Det första man möts av när man startar programmet är en inloggnings ruta där användaren skriver in ett existerande användare id(finns lagrade i en fil) och om id finns så kan man trycka på en knapp som tar en vidare till själva biblioteks delen. När man kommer till biblioteks skärmen så finns det två tabbar, en som visar vad man har lånat och sköter återlämningen och en som visar vad som kan lånas och sköter låna delen. Om man väljer att låna så måste man först välja tabben ”låna” och sedan söka på id man vill låna, alternativ att man söker på blankt och alla objekt kommer vissas. Efter det så markerar man det objektet som man vill låna och trycker på knappen ”låna”. Om man sedan vill visa vad för olika objekt man har lånat eller vill lämna tillbaks ett objekt så väljer man ”lämna tillbaka” tabben och så ser man alla sina lånade objekt. Vill man sedan lämna tillbaks ett objekt skriver man bara in id:t på det objekt som skall lämnas tillbaks och trycker på knappen ”Återlämna”

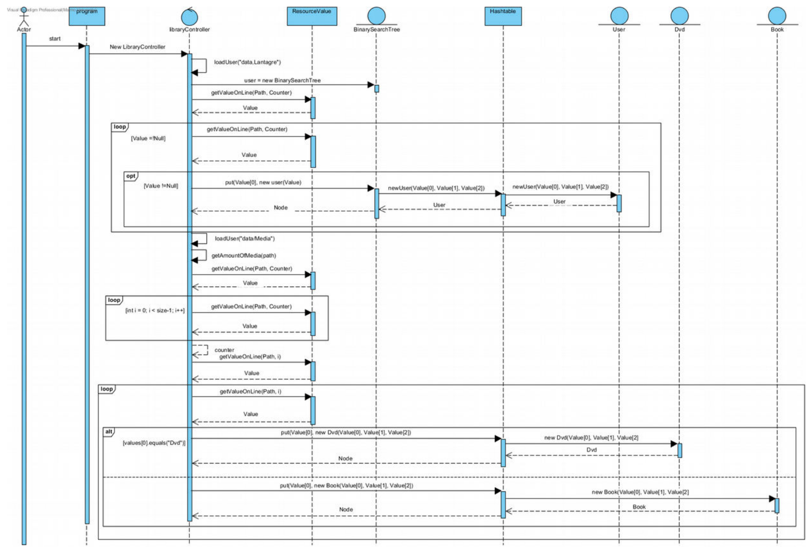
Klassdiagram



Sekvensdiagram

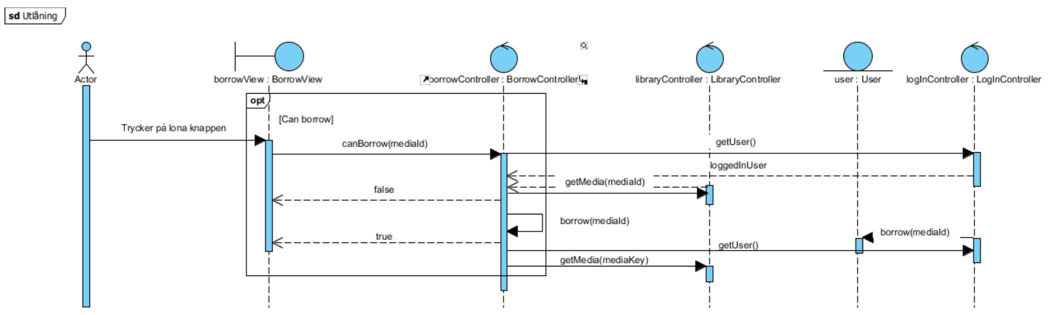
Uppstart av programmet inklusive inläsning av textfiler

Ansvarig: Rada Alasadi


Actor
program
libraryController
ResourceValue
BinarySearchTree
Hashtable
User
Dvd
Book
start
New LibraryController
loadUser("data,Lantagre")
user = new BinarySearchTree
getValueOnLine(Path, Counter)
Value
loop
[Value =!Null]
getValueOnLine(Path, Counter)
Value
opt
[Value !=Null]
put(Value[0], new user(Value)
newUser(Value[0], Value[1], Value[2])
newUser(Value[0], Value[1], Value[2])
User
User
Node
loadUser("data/Media")
getAmountOfMedia(path)
getValueOnLine(Path, Counter)
Value
loop
[int i = 0; i < size-1; i++]
getValueOnLine(Path, Counter)
Value
counter
getValueOnLine(Path, i)
Value
loop
getValueOnLine(Path, i)
Value
alt
[values[0].equals("Dvd")]
put(Value[0], new Dvd(Value[0], Value[1], Value[2])
new Dvd(Value[0], Value[1], Value[2]
Dvd
Node
put(Value[0], new Book(Value[0], Value[1], Value[2])
new Book(Value[0], Value[1], Value[2]
Book
Node

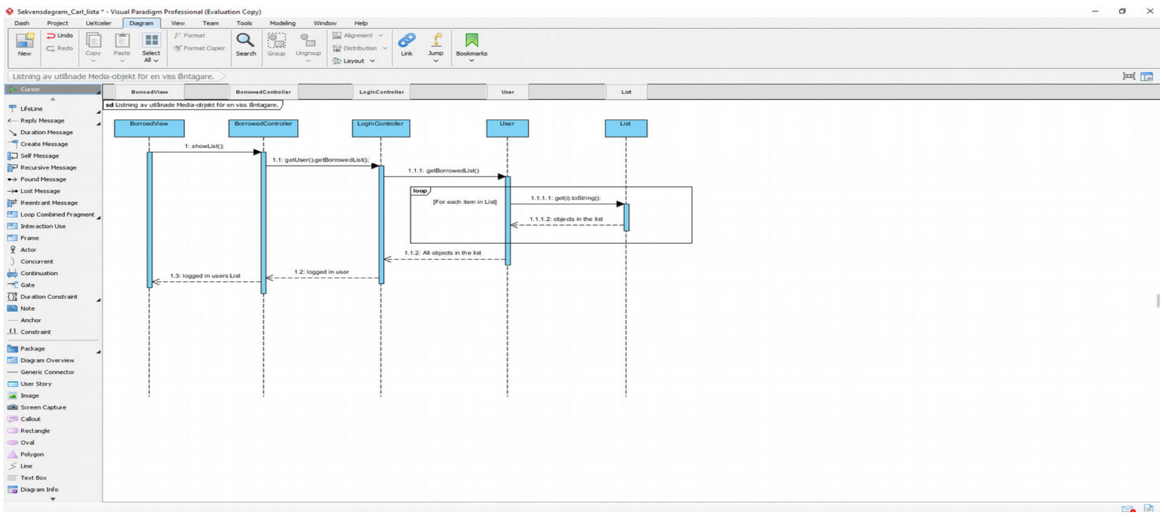
Utlåning av ett Media-objekt till en låntagare

Ansvarig: Tom Leonardsson



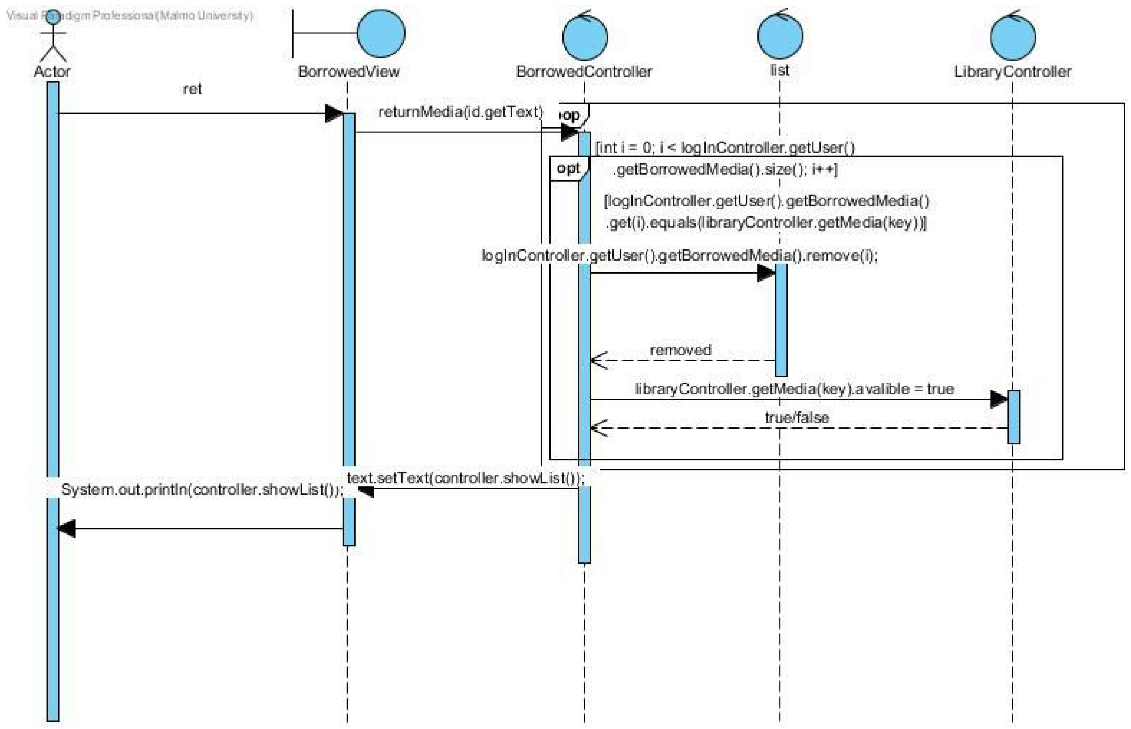
Listning av utlånade Media-objekt för en viss låntagare.

Ansvarig: Carl Zetterberg


Sekvensdagram_Carl_lista * - Visual Paradigm Professional (Evaluation Copy)
Listning av utlånade Media-objekt för en viss låntagare.
sd Listning av utlånade Media-objekt för en viss låntagare.
BorroedView
BorrowedController
LogInController
User
List
1: showList();
1.1: getUser().getBorrowedList();
1.1.1: getBorrowedList()
loop
[For each item in List]
1.1.1.1: get(i).toString();
1.1.1.2: objects in the list
1.1.2: All objects in the list
1.2: logged in user
1.3: logged in users List

Återlämning av media-objekt.

Ansvarig: David Svensson


Visual Paradigm Professional(Malmo University)
Actor
BorrowedView
BorrowedController
list
LibraryController
ret
returnMedia(id.getText)
loop
[int i = 0; i < logInController.getUser()
.getBorrowedMedia().size(); i++]
opt
[logInController.getUser().getBorrowedMedia()
.get(i).equals(libraryController.getMedia(key))]
logInController.getUser().getBorrowedMedia().remove(i);
removed
libraryController.getMedia(key).avalible = true
true/false
text.setText(controller.showList());
System.out.println(controller.showList());

Källkod

BinarySearchTree

```
package collections;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.Iterator;
import java.util.NoSuchElementException;
import collections.List;

/**
 * A binary tree that is always sorted so that it is more efficant to search in
 * @author tom.leonardsson & rolf
 *
 * @param <K>
 * @param <V>
 */
public class BinarySearchTree<K,V> {
   private Comparator<K> comparator;
   private BSTNode<K,V> tree;
   private int size;

   /**
    * create binary search tree with natrual sorting order
    */
   public BinarySearchTree() {
      comparator = new Comp();
   }

   /**
    * create tree with specific sorting order
    * @param comp the specfic sorting order
    */
   public BinarySearchTree( Comparator<K> comp ) {
      comparator = comp;
   }

   /**
    * get the root of the tree
    * @return the root-node
    */
   public BSTNode<K,V> root() {
      return tree;
   }
```

```
/**
 * get the value of the node with the specific key
 * @param the key to look after
 * @return the value of the found node
 */
public V get(K key) {
    BSTNode<K,V> node = find( key );
    if(node!=null)
        return node.value;
    return null;
}

/**
 * add node to tree with specifc value and key
 * @param value the value of the node
 * @param key the key of the node
 */
public void put(K key, V value) {
    tree = put(tree,key,value);
    size++;
}

/**
 * Remove the node with the specific key
 * @param key the key of the node to be removed
 * @return the value of the node that was removed
 */
public V remove(K key) {
    V value = get( key );
    if(value!=null) {
        tree = remove(tree,key);
        size--;
    }
    return value;
}

/**
 * check if node with key exists within the tree
 * @param key the key to look with
 * @return if the tree has the node with the key
 */
public boolean contains( K key ) {
    return find( key ) != null;
}

/**
 * get the height of the tree
 * @return the height of the tree
 */
public int height() {
    return height( tree );
}

/**
 * create an iterator from the private class Iter
 * @return an iterator of the tree
 */
public Iterator<V> iterator() {
    return new Iter();
}

private BSTNode<K,V> find(K key) {
    int res;
    BSTNode<K,V> node=tree;
    while( ( node != null ) && ( ( res = comparator.compare( key, node.key ) ) != 0 ) ) {
        if( res < 0 )
            node = node.left;
        else
            node = node.right;
    }
    return node;
}

private BSTNode<K,V> put(BSTNode<K,V> node, K key, V value) {
    if( node == null ) {
        node = new BSTNode<K,V>( key, value, null, null );
    } else {
        if(comparator.compare(key,node.key)<0) {
            node.left = put(node.left,key,value);
        } else if(comparator.compare(key,node.key)>0) {
            node.right = put(node.right,key,value);
        }
    }
    return node;
}

private BSTNode<K,V> remove(BSTNode<K,V> node, K key) {
    int compare = comparator.compare(key,node.key);
    if(compare==0) {
        if(node.left==null && node.right==null)
            node = null;
        else if(node.left!=null && node.right==null)
            node = node.left;
        else if(node.left==null && node.right!=null)
            node = node.right;
        else {
            BSTNode<K,V> min = getMin(node.right);
            min.right = remove(node.right,min.key);
            min.left = node.left;
            node = min;
        }
    } else if(compare<0) {
        node.left = remove(node.left,key);
    } else {
        node.right = remove(node.right,key);
    }
    return node;
}

private BSTNode<K,V> getMin(BSTNode<K,V> node) {
    while(node.left!=null)
        node = node.left;
    return node;
}

private int height( BSTNode<K,V> node ) {
    if( node == null )
        return -1;
    return 1 + Math.max( height( node.left ), height( node.right ));
}

/**
 * Get the size of the tree
 * @return the size of the tree
 */
public int size() {
    return size;
}

/**
 * get the size of the tree
 * @return the size of the tree
 */
public int size1() {
        return (tree != null) ? this.tree.size() : 0;
}

/**
 * get the size of the tree
 * @return the size of the tree
 */
public int size2() {
        return (tree != null) ? 1 + ((tree.right != null) ? tree.right.size() : 0) + ((tree.left != null) ? tree.left.size() : 0) : 0;
}
```

```
/**
 * get the first value in the tree, which will always be on the left since the tree is always sorted
 * @return the first value in the tree
 */
public V first(){
        if(tree == null) return null;
        BSTNode<K, V> node = tree;
        while(node.left != null)
                node = node.left;
   return node.value;
}

/**
 * get the last value in the tree, which will always be on the right since the tree is always sorted
 * @return the last value in the tree
 */
public V last(){
        if(tree == null) return null;
        BSTNode<K, V> node = tree;
        while(node.right != null)
                node = node.right;
   return node.value;
}

/**
 * print the contents of the tree in preorder
 */
public void printPreorder() {
        printPreorder(tree);
}

private void printPreorder(BSTNode<K,V> node) {
        if(node != null) {
                System.out.println("key: " + node.key + " | value: " + node.value);
                printPreorder(node.left);
                printPreorder(node.right);
        }
}

/**
 * Print out the contents of the tree
 */
public void print() {
        print(tree);
}
```

```
private void print(BSTNode<K,V> node) {
        if(node != null) {
                print(node.left);
                System.out.println("key: " + node.key + " | value: " + node.value);
                print(node.right);
        }
}

/**
 * get a list of the keys of the nodes in the tree
 * @return a list of keys
 */
public ArrayList<K> keys(){
        ArrayList<K> list = new ArrayList<K>();
        keys(tree, list);
    return list;
}

/**
 * Put all the keys into a list recusivly
 * @param node the current nodes key being put in
 * @param list the list that is being filled with keys
 */
private void keys(BSTNode<K,V> node, ArrayList<K> list){
        if(node!=null) {
                keys(node.left, list);
      list.add(node.key);
      keys(node.right, list);
    }
}

/**
 * get a list filled with the values of all the nodes
 * @return a list with all the values of the trees nodes
 */
public ArrayList<V> values(){
        if(tree == null) return null;
        Iterator elements = iterator();
        ArrayList<V> tmp = new ArrayList<V>();
        while(elements.hasNext())
                tmp.add((V) elements.next());
        return tmp;
}

/**
 * Sorting order for the tree
 * @author Rolf Axelsson
 *
 */
```

```
private class Comp implements Comparator<K> {
            /**
             * Get if the key is greater, lesser or equal to the other
             * @return if the key is greater, lesser or equal to the other
             */
    public int compare( K key1, K key2 ) {
        Comparable<K> k1 = ( Comparable<K> )key1;
        return k1.compareTo( key2 );
    }
}

/**
 * Iterator for the tree
 * @author Rolf Axelsson
 *
 */
private class Iter implements Iterator<V> {
    ArrayList<V> list = new ArrayList<V>();
    int index = -1;

    /**
     * Create iterator
     */
    public Iter() {
        inOrder(tree);
    }

    /**
     * Put all the nodes into a list recursivly
     * @param node the node that is being put in
     */
    private void inOrder(BSTNode<K,V> node) {
        if(node!=null) {
            inOrder(node.left);
            list.add(node.value);
            inOrder(node.right);
        }
    }

    /**
     * the iterator has a next value
     * @return if there is a next value
     */
    public boolean hasNext() {
        return index<list.size()-1;
    }

    /**
```

```
         * get the next value in the list
         * @return the next value
         */
        public V next() {
            if(!hasNext())
                throw new NoSuchElementException();
            index++;
            return list.get(index);
        }

        /**
         * throws exception
         */
        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
```

Book

```
package collections;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.Iterator;
import java.util.NoSuchElementException;
import collections.List;

/**
 * A binary tree that is always sorted so that it is more efficant to search in
 * @author tom.leonardsson & rolf
 *
 * @param <K>
 * @param <V>
 */
public class BinarySearchTree<K,V> {
    private Comparator<K> comparator;
    private BSTNode<K,V> tree;
    private int size;

    /**
     * create binary search tree with natrual sorting order
     */
    public BinarySearchTree() {
        comparator = new Comp();
    }

    /**
     * create tree with specific sorting order
     * @param comp the specfic sorting order
     */
    public BinarySearchTree( Comparator<K> comp ) {
        comparator = comp;
    }

    /**
     * get the root of the tree
     * @return the root-node
     */
    public BSTNode<K,V> root() {
        return tree;
    }

    /**
     * get the value of the node with the specific key
     * @param the key to look after
     * @return the value of the found node
     */
    public V get(K key) {
        BSTNode<K,V> node = find( key );
        if(node!=null)
            return node.value;
        return null;
    }

    /**
     * add node to tree with specifc value and key
     * @param value the value of the node
     * @param key the key of the node
     */
    public void put(K key, V value) {
        tree = put(tree,key,value);
        size++;
    }

    /**
     * Remove the node with the specific key
     * @param key the key of the node to be removed
     * @return the value of the node that was removed
     */
    public V remove(K key) {
        V value = get( key );
        if(value!=null) {
            tree = remove(tree,key);
            size--;
        }
        return value;
    }

    /**
     * check if node with key exists within the tree
     * @param key the key to look with
     * @return if the tree has the node with the key
     */
    public boolean contains( K key ) {
        return find( key ) != null;
    }

    /**
     * get the height of the tree
     * @return the height of the tree
     */
    public int height() {
        return height( tree );
    }

    /**
 * create an iterator from the private class Iter
 * @return an iterator of the tree
 */
public Iterator<V> iterator() {
    return new Iter();
}

private BSTNode<K,V> find(K key) {
    int res;
    BSTNode<K,V> node=tree;
    while( ( node != null ) && ( ( res = comparator.compare( key, node.key ) ) != 0 ) ) {
        if( res < 0 )
            node = node.left;
        else
            node = node.right;
    }
    return node;
}

private BSTNode<K,V> put(BSTNode<K,V> node, K key, V value) {
    if( node == null ) {
        node = new BSTNode<K,V>( key, value, null, null );
    } else {
        if(comparator.compare(key,node.key)<0) {
            node.left = put(node.left,key,value);
        } else if(comparator.compare(key,node.key)>0) {
            node.right = put(node.right,key,value);
        }
    }
    return node;
}

private BSTNode<K,V> remove(BSTNode<K,V> node, K key) {
    int compare = comparator.compare(key,node.key);
    if(compare==0) {
        if(node.left==null && node.right==null)
            node = null;
        else if(node.left!=null && node.right==null)
            node = node.left;
        else if(node.left==null && node.right!=null)
            node = node.right;
        else {
            BSTNode<K,V> min = getMin(node.right);
            min.right = remove(node.right,min.key);
            min.left = node.left;
            node = min;
        }
    } else if(compare<0) {
        node.left = remove(node.left,key);
    } else {
        node.right = remove(node.right,key);
    }
    return node;
}

private BSTNode<K,V> getMin(BSTNode<K,V> node) {
    while(node.left!=null)
        node = node.left;
    return node;
}

private int height( BSTNode<K,V> node ) {
    if( node == null )
        return -1;
    return 1 + Math.max( height( node.left ), height( node.right ));
}

/**
 * Get the size of the tree
 * @return the size of the tree
 */
public int size() {
    return size;
}

/**
 * get the size of the tree
 * @return the size of the tree
 */
public int size1() {
                                return (tree != null) ? this.tree.size() : 0;
}

/**
 * get the size of the tree
 * @return the size of the tree
 */
public int size2() {
                                return (tree != null) ? 1 + ((tree.right != null) ? tree.right.size() : 0) + ((tree.left != null) ? tree.left.size() : 0) : 0;
}

/**
 * get the first value in the tree, which will always be on the left since the tree is always sorted
 * @return the first value in the tree
 */
public V first(){
                                if(tree == null) return null;
                                BSTNode<K, V> node = tree;
                                while(node.left != null)
                                                node = node.left;
    return node.value;
}

/**
 * get the last value in the tree, which will always be on the right since the tree is always sorted
 * @return the last value in the tree
 */
public V last(){
                    if(tree == null) return null;
                    BSTNode<K, V> node = tree;
                    while(node.right != null)
                                        node = node.right;
    return node.value;
}

/**
 * print the contents of the tree in preorder
 */
public void printPreorder() {
                    printPreorder(tree);
}

private void printPreorder(BSTNode<K,V> node) {
                    if(node != null) {
                                        System.out.println("key: " + node.key + " | value: " + node.value);
                                        printPreorder(node.left);
                                        printPreorder(node.right);
                    }
}

/**
 * Print out the contents of the tree
 */
public void print() {
                    print(tree);
}

private void print(BSTNode<K,V> node) {
                    if(node != null) {
                                        print(node.left);
                                        System.out.println("key: " + node.key + " | value: " + node.value);
                                        print(node.right);
                    }
}

/**
 * get a list of the keys of the nodes in the tree
 * @return a list of keys
 */
public ArrayList<K> keys(){
                    ArrayList<K> list = new ArrayList<K>();
                    keys(tree, list);
    return list;
}

/**
 * Put all the keys into a list recusivly
 * @param node the current nodes key being put in
 * @param list the list that is being filled with keys
 */
private void keys(BSTNode<K,V> node, ArrayList<K> list){
                    if(node!=null) {
                                        keys(node.left, list);
        list.add(node.key);
        keys(node.right, list);
    }
}

/**
 * get a list filled with the values of all the nodes
 * @return a list with all the values of the trees nodes
 */
public ArrayList<V> values(){
                    if(tree == null) return null;
                    Iterator elements = iterator();
                    ArrayList<V> tmp = new ArrayList<V>();
                    while(elements.hasNext())
                                        tmp.add((V) elements.next());
                    return tmp;
}

/**
 * Sorting order for the tree
 * @author Rolf Axelsson
 *
 */
private class Comp implements Comparator<K> {
                    /**
                     * Get if the key is greater, lesser or equal to the other
                     * @return if the key is greater, lesser or equal to the other
                     */
    public int compare( K key1, K key2 ) {
        Comparable<K> k1 = ( Comparable<K> )key1;
        return k1.compareTo( key2 );
    }
}

/**
 * Iterator for the tree
 * @author Rolf Axelsson
 *
 */
```

```
    private class Iter implements Iterator<V> {
        ArrayList<V> list = new ArrayList<V>();
        int index = -1;

        /**
         * Create iterator
         */
        public Iter() {
            inOrder(tree);
        }

        /**
         * Put all the nodes into a list recursivly
         * @param node the node that is being put in
         */
        private void inOrder(BSTNode<K,V> node) {
            if(node!=null) {
                inOrder(node.left);
                list.add(node.value);
                inOrder(node.right);
            }
        }

        /**
         * the iterator has a next value
         * @return if there is a next value
         */
        public boolean hasNext() {
            return index<list.size()-1;
        }

        /**
         * get the next value in the list
         * @return the next value
         */
        public V next() {
            if(!hasNext())
                throw new NoSuchElementException();
            index++;
            return list.get(index);
        }

        /**
         * throws exception
         */
        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
```

BSTNode

```java
package collections;

/**
 * Node for the binary search treee, has a key to be indentfied with, a value and has a node on both sides
 * @author rolf
 *
 * @param <K>
 * @param <V>
 */
class BSTNode<K,V> {
    K key;
    V value;
    BSTNode<K,V> left;
    BSTNode<K,V> right;

    /**
     * Create a node with a specific key, value, left and right node
     * @param key the key of the node
     * @param value the value of the node
     * @param left the left node of the node
     * @param right the right node of the node
     */
    public BSTNode( K key, V value, BSTNode<K,V> left, BSTNode<K,V> right ) {
        this.key = key;
        this.value = value;
        this.left = left;
        this.right = right;
    }

    /**
     * get the height of the tree
     * @return the height of the tree
     */
    public int height() {
        int leftH = -1, rightH = -1;
        if( left != null )
            leftH = left.height();
        if( right != null )
            rightH = right.height();
        return 1 + Math.max( leftH, rightH );
    }

    /**
     * Get the size of the tree
     * @return the size of the tree
     */
    public int size() {
        int leftS = 0, rightS = 0;
        if( left != null )
            leftS = left.size();
        if( right != null )
            rightS = right.size();
        return 1 + leftS + rightS;
    }

    /**
     * print the contents of the tree
     */
    public void print() {
        if( left != null)
            left.print();
        System.out.println(key + ": " + value);
        if( right != null )
            right.print();
    }

    public void showTree() {
    //  javax.swing.JOptionPane.showMessageDialog( null, new ShowBST<K,V>( this, 800,600 ), "Show tree", javax.swing.JOptionPane.PLAIN_MESSAGE );
    }
}
```

Bucket

```
package collections;

/**
 * Holds a key and a value, also what state it is in
 * @author tom.leonardsson
 *
 * @param <K>
 * @param <V>
 */
public class Bucket<K, V> {
			enum State { EMPTY, OCCUPIED, REMOVED };

			public State state;

			public K key;
			public V value;
}
```

Dvd

```
package collections;
/**
 *
 * Creates an Dvd object that extends Media
 *
 *
 * @author Carl Zetterberg
 *
 */
public class Dvd extends Media {
	/**
	 * Instans variables
	 */
	private String[] actors;

	/**
	 * Contructor that make a dvd object
	 * @param id of the object
	 * @param tite on the object
	 * @param year the dvd was realesed
	 * @param actors that stared the dvd
	 */
	public Dvd(String id, String tite, int year, String[] actors) {
		super(id, tite, year);
		this.actors = actors;
	}

	/**
	 * Get the actor that stared in the dvd
	 *
	 * @return and array of the actors
	 */
	public String[] getActors(){
		return actors;
	}
	/**
	 * An toString that decsribes the object
	 *
	 * @return a string that describes the object
	 */
	public String toString() {
		String tmp = "{";
		tmp += getId() + ", " + getTitle() + ", " + getYear() + ", ";
		for(int i = 0; i < actors.length; i++)
			tmp += actors[i] + ((i < actors.length-1) ? ", " : "");
		return tmp + "}";
	}
}
```

Hashable

```
package collections;

import java.util.ArrayList;
import java.util.Iterator;

import collections.Bucket.State;

/**
 * Table of values that are reached with a hashindex based on the key. O(1) search
 * @author tom.leonardsson
 *
 * @param <K> key
 * @param <V> value
 */
public class Hashtable<K, V> {
	private Bucket<K, V> table[];

	private int size;

	/**
	 * Create empty hashtable with specifc capacity
	 * @param capacity the specifc start capacity
	 */
	public Hashtable(int capacity) {
		table = (Bucket<K, V>[])new Bucket[capacity*4];
		for(int i = 0; i < table.length; i++) {
			table[i] = new Bucket<K, V>();
		}
	}


	/**
	 * Create hashIndex based on the keys hashcode
	 * @param key the key to get the index of
	 * @return the index in the table of the key
	 */
	public int hashIndex(K key) {
		int hashCode = key.hashCode();
		hashCode = hashCode % table.length;
		return (hashCode < 0) ? -hashCode : hashCode;
	}

	/**
	 * Put a new element with a key and a value into the table
	 * @param key the key of the element
	 * @param value the value of the element
	 */
	public void put(K key, V value) {
		int counter = 0;
		int removed = -1;
		int hashIndex = hashIndex(key);

		while(counter < table.length && table[hashIndex].state != State.EMPTY && table[hashIndex].key != key) {
			if(removed == -1 && table[hashIndex].state != State.REMOVED)
				removed = hashIndex;

			counter += 1;
			hashIndex += 1;

			hashIndex = (hashIndex == table.length) ? 0 : hashIndex;
		}

		if(removed != -1) hashIndex = removed;

		table[hashIndex].key = key;
		table[hashIndex].value = value;
		table[hashIndex].state = State.OCCUPIED;
		size += 1;
	}

	/**
	 * Remove a specifc element from the table with a key
	 * @param key the key to look for
	 * @return the removed element, null if not found
	 */
	public V remove(K key) {
		int counter = 0;
		int hashIndex = hashIndex(key);

		if(get(key) != null) {
			while(counter < table.length && !key.equals(table[hashIndex].key)) {
				counter += 1;
				hashIndex += 1;
				hashIndex = (hashIndex == table.length) ? 0 : hashIndex;
			}

			Bucket<K, V> tmp = table[hashIndex];

			table[hashIndex].key = null;
			table[hashIndex].value = null;
			table[hashIndex].state = State.REMOVED;
			size -= 1;

			return tmp.value;
		}
		return null;
	}
```

```
                        /**
                         * get the value of an element with specifc key
                         * @param key the key to look for
                         * @return the value of the element with the key or null if not found
                         */
                        public V get(K key) {
                                    int counter = 0;
                        int hashIndex = hashIndex(key);

                        while(counter < table.length && table[hashIndex].state != State.EMPTY && !key.equals(table[hashIndex ].key)) {
                                    counter += 1;
                                    hashIndex += 1;
                                    hashIndex = (hashIndex == table.length) ? 0 : hashIndex;
                        }

  return key.equals(table[hashIndex].key) ? table[hashIndex].value : null;
                        }

                        /**
                         * Create iterator that iterates over the values in the hashtable
                         * @return the iterator
                         */
                        public Iterator<V> values() {
                                    ArrayList<V> l = new ArrayList<V>();
                                    for(int i = 0; i < table.length; i++) {
                                                if(table[i].state == State.OCCUPIED) {
                                                            l.add(table[i].value);
                                                }
                                    }

                                    return l.iterator();
                        }

                        /**
                         * Create iterator that iterates over the keys in the hashtable
                         * @return the iterator
                         */
                        public Iterator<K> keys() {
                                    ArrayList<K> l = new ArrayList<K>();
                                    for(int i = 0; i < table.length; i++) {
                                                if(table[i].state == State.OCCUPIED) {
                                                            l.add(table[i].key);
                                                }
                                    }
                                    return l.iterator();
                        }

                        /**
                         * Create structured string that shows the contents of the table
                         * @return the structured string
                         */
                        public String toString() {
                                    String tmp = "";
                                    for(int i = 0; i < table.length; i++) {
                                                tmp += (table[i].key + " - " + table[i].value) + ((i < table.length-1) ? "\n " : "");
                                    }
                                    return tmp;
                        }
}
```

List

```
package collections;

/**
 * List that has a generic array
 * @author tom.leonardsson
 *
 * @param <E>
 */
public class List<E> {
		private E[] elements;

		/**
		 * Create an empty list
		 */
		public List() {
			elements = (E[])new Object[0];
		}

		/**
		 * Add new element into the list and make it one element bigger
		 * @param e the element to put in
		 * @return the element that has been added
		 */
		public E add(E e) {
			E[] tmp = (E[])new Object[size()+1];

			for(int i = 0; i < size(); i++)
				tmp[i] = elements[i];
			tmp[size()] = e;
			elements = tmp;
			return e;
		}

		/**
		 * Remove an element from the list
		 * @param index the position of the element in the list
		 * @return the removed element
		 */
		public E remove(int index) {
			E[] tmp = (E[])new Object[size()-1];

			int offset = 0;

			for(int i = 0; i < size(); i++) {
				if(i != index) {
					tmp[i-offset] = elements[i];
				} else {
					offset += 1;
				}
			}

			E removed = elements[index];

			elements = tmp;

			return removed;
		}

		/**
		 * Get the index of the element and remove it based on it's index
		 * @param e the element to look for
		 * @return the removed element
		 */
		public E remove(E e) {
			return remove(getIndex(e));
		}

		/**
		 * Remove the first element in the list
		 * @return the removed element
		 */
		public E removeFirst() {
			return remove(0);
		}

		/**
		 * Remove everything by removing the first element until it's empty
		 */
		public void clear() {
			while(size() > 0)
				removeFirst();
		}

		/**
		 * Get the index of a specifc element
		 * @param e the element to look for
		 * @return the index of the element, -1 if not found
		 */
		public int getIndex(E e) {
			for(int i = 0; i < size(); i++)
				if(elements[i].equals(e)) return i;
			return -1;
		}

		/**
		 * Get the element at a specifc index
		 * @param index the index to get
		 * @return the element in the index
		 */
		public E get(int index) {
				return elements[index];
		}

		/**
		 * Get the size of the list
		 * @return the size of the list
		 */
		public int size() {
				return elements.length;
		}

		/**
		 * Create a structred string that shows the contents of the list
		 * @return the structured string
		 */
		public String toString() {
				String tmp = "{";
				for(int i = 0; i < size(); i++) {
						tmp += elements[i] + ((i < size()-1) ? ", " : "");
				}
				return tmp + "}";
		}
}
```

Media

```
package collections;
/**
 *
 * Creates an abstact media class that Book and Dvd will extend
 *
 * @author Carl Zetterberg
 *
 */
public abstract class Media {
				/**
				 * Instans variables
				 */
				private String id;
				private String title;

				private int year;

				public boolean avalible;
				/**
				 * constructor that takes a id title and year and sets it to available
				 * @param id for the Media object
				 * @param title o the object
				 * @param year that the object was released
				 */
				public Media(String id, String title, int year) {
								this.id = id;
								this.title = title;
								this.year = year;

								avalible = true;
				}
				/**
				 * Gets the objects id
				 *
				 * @return id of the object
				 */
				public String getId() {
								return id;
				}
				/**
				 * Gets the objects year it was released
				 *
				 * @return year of the object
				 */
				public int getYear(){
								return year;
				}
				/**
				 * Gets the objects title
				 *
				 * @return title of the object
				 */
				public String getTitle(){
								return title;
				}
				/**
				 * Compare to objects id
				 *
				 * @param obj the object you want to compare with
				 * @return true if the id match , else false
				 */

				public boolean equals( Object obj ) {
								if(obj instanceof Media) {
												Media media = (Media)obj;
												return id.equals( media.getId() );
								}
								return false;
				}
}
```

ResourceReader

```
package collections;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

/**
 * Class with one method to load data from files
 * @author tom.leonardsson
 *
 */
public class ResourceReader {
		/**
		 * Get the elements on a specifc line in a file at a specif path, split at every ";"
		 * @param path the path to the file
		 * @param line the line to read
		 * @return an array of strings that holds every piece of the line
		 * @throws IOException
		 */
		public static String[] getValuesOnLine(String path, int line) throws IOException {
				String[] tmp = null;
				String l = "";

				try {
						BufferedReader br = new BufferedReader(new FileReader(path
+ ".txt"));

						int count = 0;

						l = br.readLine();

						while(l != null) {
								if(count == line) break;
								l = br.readLine();
								count += 1;
						}
						br.close();
				} catch (FileNotFoundException e) {
						e.printStackTrace();
				}

				if(l != null)
						tmp = l.split(";");

				return tmp;
		}
}
```

User

```
package collections;

/**
 * An entity class that keeps track of a user
 * @author tom.leonardsson & David Svensson
 *
 */
public class User {
		private String id;
		private String name;
		private String phoneNumber;
		private List<Media> borrowedMedia;

		/**
		 * Create user with specifc id, name and phonenumber
		 * @param id the id of the user
		 * @param name the name of the user
		 * @param phoneNumber the phone number of the user
		 */
		public User(String id, String name, String phoneNumber) {
				this.id = id;
				this.name = name;
				this.phoneNumber = phoneNumber;

				borrowedMedia = new List<Media>();
		}

		/**
		 * Get the list of what has been borrowed
		 * @return the list of what has been borrowed
		 */
		public List<Media> getBorrowedMedia() {
				return borrowedMedia;
		}

		/**
		 * Set the phone number
		 * @param phonenumber
		 */
		public void setPhoneNumber(String phonenumber){
				this.phoneNumber=phonenumber;
		}
		/**
		 * Get the name of the user
		 * @return the name of the user
		 */
		public String getName(){
				return this.name;
		}

		/**
		 * Get the id of the user
		 * @return the id of the user
		 */
		public String getId(){
				return this.id;
		}

		/**
		 * Add a media object to the list of borrowed media
		 * @param the media to add
		 */
		public void borrow(Media media){
				borrowedMedia.add(media);
				System.out.println(getBorrowedList());
		}

		/**
		 * Get a structured string of the list of media that has been borrowed
		 * @return the strucuterd string
		 */
		public String getBorrowedList() {
				String tmp = "";

				for(int i = 0; i < borrowedMedia.size(); i++)
						tmp += borrowedMedia.get(i).toString() + "\n";

				return tmp;
		}

		/**
		 * Create a structured string of the contents of the user
		 * @return the structured string
		 */
		public String toString() {
				return "{" + id + ", " + name + ", " + phoneNumber + "}";
		}
}
```

BorrowController

```
package library;

import java.util.Iterator;

import collections.List;
/**
 * This is a controller for the borrow funkction. It test first if its possible
 * for user to borrow and if its true, then it proceeds to make the borrow.
 * there is a search here for media, it uses the keys to identify them and puts them on display.
 * @author Tom Leonardsson, Murtadha alasadi
 *
 */
public class BorrowController {
		/**
		 * instance variables
		 */
		private LibraryController libraryController;
		private LogInController logInController;

		/**
		 * this is the constructor for the class. it has two parameters for
		 * the other controllers which are needed here.
		 * @param libraryController
		 * @param logInController
		 */
		public BorrowController(LibraryController libraryController, LogInController logInController) {
			this.libraryController = libraryController;
			this.logInController = logInController;
		}
		/**
		 * This is a method that test if its possible to borrow a  media,
		 * if its possible, it returns true if its possible, otherwise
		 * it return false.
		 * @param key
		 * @return return true or false.
		 */
		public boolean canBorrow(String key) {
			return logInController.getUser() != null && libraryController.getMedia(key).avalible;
		}
		/**
		 * this is the method that makes the borrow. Firstly it get the key for
		 * the media and after that goes and puts the media under the specified user.
		 * @param key
		 */
		public void borrow(String key) {
			System.out.println(libraryController.getMedia(key));
			logInController.getUser().borrow(libraryController.getMedia(key));
			libraryController.getMedia(key).avalible = false;
		}

		/**
		 * check if media exists
		 * @return if it exists
		 */
		public boolean mediaExists(String key) {
			return libraryController.getMedia(key) != null;
		}

		/**
		 * this is a search engine for the list of the media.
		 * after you type a certain key, and loops the whole list, and delete
		 * all the media that doesnt match the the key.
		 * @param search
		 * @param t
		 */
		public void searchList(String search, java.awt.List t) {
			t.removeAll();

			Iterator iter = libraryController.getMedia().keys();

			while(iter.hasNext()) {
				String n = iter.next().toString();
				if(n.contains(search)) {
					t.add(n);
				}
			}
		}
}
```

BorrowedController

```
package library;
/**
 * this the window for the borrowed function.
 * @author Tom Leonardsson, Murtadha Alasadi
 *
 */
public class BorrowedController {
			/**
			 * instance veriables
			 */
			private LogInController logInController;
			private LibraryController libraryController;
			/**
			 * the constructor for the class
			 * @param libraryController
			 * @param logInController
			 */
			public BorrowedController(LibraryController libraryController, LogInController
logInController) {
						this.logInController = logInController;
						this.libraryController = libraryController;
			}
			/**
			 * this the method that return a certain media. it uses the key which is
			 * a parameter. it gets the user that wants to return the media. when its find, its sets
			 * the availability for the media.
			 * @param key
			 */
			public void returnMedia(String key) {
						for(int i = 0; i < logInController.getUser().getBorrowedMedia().size(); i++) {

if(logInController.getUser().getBorrowedMedia().get(i).equals(libraryController.getMedia(key))) {

			logInController.getUser().getBorrowedMedia().remove(i);
												libraryController.getMedia(key).avalible =
true;
									}
						}
			}
			/**
			 * this method return the list of all the media that a certain user have.
			 * @return return the media as a string.
			 */

			public String showList() {
						return logInController.getUser().getBorrowedList();
			}
}
```

BorrowedView

```
package library;

import java.awt.*;

import javax.swing.*;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This is the window for the borrow panel, here you choose between all the
 * medias and then borrow one, after that your borrowed object will be put in
 * your borrowed list.
 *
 * @author Murtadha Al-asadi
 *
 */

public class BorrowedView extends JPanel {
	/**
	 * instance variables for the program
	 */
	private BorrowedController controller;

	private JButton ret = new JButton("Återlämna");
	private JButton logout = new JButton("Logga ut");

	private JTextArea text = new JTextArea();
	private JTextField id = new JTextField();

	private JPanel btnPan = new JPanel();
	private JPanel view = new JPanel();
	private JPanel viewer = new JPanel();

	private JLabel mediaId = new JLabel("ID:");
	private JLabel borrowed = new JLabel("Utlånade:");

	/**
	 * this is the constructor, here are all the objects for the panel are
	 * written and drawn
	 *
	 * @param controller
	 */

	public BorrowedView(BorrowedController controller) {
		this.controller = controller;

		setLayout(new BorderLayout());
		btnPan.setLayout(new GridLayout(10, 0));
		view.setLayout(new GridLayout(0, 2));

		mediaId.setPreferredSize(new Dimension(80, mediaId.getSize().height));
		borrowed.setPreferredSize(new Dimension(80, borrowed.getSize().height));

		ButtonListener l = new ButtonListener();
		ret.addActionListener(l);
		logout.addActionListener(l);

		btnPan.add(mediaId);
		btnPan.add(id);
		btnPan.add(ret);
		btnPan.add(logout);

		view.add(borrowed);
		viewer.add(view);
		viewer.add(text, BorderLayout.WEST);

		add(viewer);
		add(btnPan, BorderLayout.EAST);

		this.add(viewer);

		this.add(view, BorderLayout.NORTH);
		this.add(text, BorderLayout.WEST);
		this.add(btnPan, BorderLayout.EAST);
	}

	/**
	 * here are the list of borrowed media. this method update the
	 * list after one borrow is made.
	 */

	public void setTextToBorrowedList() {
		text.setText(controller.showList());
	}

	/**
	 * this is the inner class for the actionlistener for the buttons to work.
	 * ever button have an action to do when pressed
	 *
	 * @author Murtadha alasadi
	 *
	 */

	public class ButtonListener implements ActionListener {
		public void actionPerformed(ActionEvent e) {
			if (e.getSource() == ret) {
				controller.returnMedia(id.getText());
				text.setText(controller.showList());
				System.out.println(controller.showList());
			}
			if (e.getSource() == logout) {

			}
		}
	}
}
```

BorrowView

```
package library;

import java.awt.*;
import javax.swing.*;


import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

/**
 * This is the window for the borrow panel, here you choose between all the
 * medias and then borrow one, after that your borrowed object will be put in
 * your borrowed list.
 *
 * @author Murtadha Al-asadi
 *
 */
public class BorrowView extends JPanel {
                    /**
                     * instance variables
                     */
                    private BorrowController controller;

                    private JTextField mediaId = new JTextField();
                    private JTextField sok = new JTextField();

                    private JLabel mId = new JLabel("Media ID: ");
                    private JLabel sokf = new JLabel("Sök");

                    private List list = new List();

                    private JPanel lonField = new JPanel();
                    private JPanel btnField = new JPanel();

                    private JButton lona = new JButton("Låna");
                    private JButton logout = new JButton("logga ut");
                    private JButton sokMed = new JButton("Sök");

                    /**
                     * this is the constructor, here are all the objects for the panel are
                     * written and drawn
                     *
                     * @param controller
                     */

                    public BorrowView(BorrowController controller) {
                                        this.controller = controller;

                                        setLayout(new BorderLayout());
                                        lonField.setLayout(new GridLayout(2, 6));
                                        btnField.setLayout(new GridLayout(10, 0));

                                        mId.setPreferredSize(new Dimension(80, mId.getSize().height));
                                        sokf.setPreferredSize(new Dimension(80, sokf.getSize().height));

                                        ButtonListener l = new ButtonListener();
                                        sokMed.addActionListener(l);
                                        lona.addActionListener(l);
                                        logout.addActionListener(l);

                                        btnField.add(sokMed);
                                        btnField.add(lona);
                                        btnField.add(logout);


                                        sok.setToolTipText("Mata in media-ID");
                                        mediaId.setToolTipText("Mata in media -ID");

                                        lonField.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
                                        lonField.add(sok);
                                        lonField.add(sokf);
                                        lonField.add(mediaId);
                                        lonField.add(mId);

                                        this.add(btnField, BorderLayout.EAST);
                                        this.add(lonField, BorderLayout.NORTH);
                                        this.add(list, BorderLayout.WEST);

                                        ItemChangeListener itemListerner = new ItemChangeListener();

                                        list.addItemListener(itemListerner);
                    }
                    /**
                     * here are the itemlister for the list that we implement.
                     * this is for the list to update and after a borrow i done.
                     * @author Tom Leonardsson
                     *
                     */

                    class ItemChangeListener implements ItemListener{
                        public void itemStateChanged(ItemEvent event) {
                                        if(event.getSource() == list) {
                                                            mediaId.setText(list.getSelectedItem());
                                        }
                        }
                    }
```

```
			/**
			 * this is the buttonlistner for all the buttons.
			 * every button have an action to be made when it will
			 * be pressed.
			 * @author Murtadha alasadi
			 *
			 */

			public class ButtonListener implements ActionListener {
				public void actionPerformed(ActionEvent e) {
					if (e.getSource() == sokMed) {
						controller.searchList(sok.getText(), list);
					}

					if (e.getSource() == lona) {
						if(controller.mediaExists(mediaId.getText()) &&
controller.canBorrow(mediaId.getText())) {
							controller.borrow(mediaId.getText());
						}
					}
					if (e.getSource() == logout) {
					}
				}
			}
}
```

LibraryController

```
package library;

import java.io.IOException;

import javax.swing.JPanel;

import collections.ResourceReader;
import collections.Book;
import collections.Dvd;
import collections.Hashtable;
import collections.Media;
import collections.User;
import collections.BinarySearchTree;

/**
 * this is the "main" controller for the whole program. Here is
 * pretty much every logic and instance for the program to start
 * and load the users and the media.
 * @author Tom Leonardsson
 *
 */
public class LibraryController {
		/**
		 * instance variables.
		 */
		private Hashtable<String, Media> media;
		private BinarySearchTree<String, User> users;

		private LogInController logInController;

		private LibraryView view;

		public LibraryController() {
				loadUsers("data/Lantagare");
				loadMedia("data/Media");
		}
		/**
		 * this is a set-method for instancing the logInCtroller.
		 * @param logInController
		 */
		public void setLogInController(LogInController logInController) {
				this.logInController = logInController;
		}
		/**
		 * this is the get-method, upon call this method returns the main view.
		 * @return view
		 */
		public LibraryView getView() {
				return view;
		}
		/**
		 * a set-method for changing the view if needed.
		 * @param view
		 */
		public void setView(LibraryView view) {
				this.view = view;
		}
		/**
		 * this is a get-method to return the media of type
		 * hashtable.
		 * @return Hashtable
		 */
		public Hashtable<String, Media> getMedia() {
				return media;
		}
		/**
		 * this is a get-method that return the user when called.
		 * @return user
		 */

		public User getLoggedInUser() {
				return logInController.getUser();
		}
		/**
		 * this is a get-method with a key as a parameter. this return the key
		 * of the user when called upon
		 * @param key
		 * @return
		 */
		public User getUser(String key) {
				return users.get(key);
		}
		/**
		 * this a get-method with a key as a parameter. this returns
		 * the key of the media when called upon
		 * @param key
		 * @return return the key
		 */

		public Media getMedia(String key) {
				return media.get(key);
		}
		/**
		 * This method is for loading the users to the database. first of all
		 * it as a parameter 'path'. this is the source file where all users
		 * are. it then create a new BinarySearchTree. after that it get the first
		 * value of a user, and then goes into a loop for gathering all the
		 * users in the file. for every information in get, it create
	 * a new user object. It keeps running till all users are gathered.
	 * there is an IOException, if it couldnt find any user in the file.
	 * @param path
	 */

	public void loadUsers(String path) {
		users = new BinarySearchTree<String, User>();
		String[] values = new String[0];
		int counter = 0;
		try {
			values = ResourceReader.getValuesOnLine(path, counter);
			while(values != null) {
				values = ResourceReader.getValuesOnLine(path, counter);
				if(values != null) users.put(values[0], new User(values[0], values[1],
values[2]));

				counter += 1;
			}
		} catch (IOException e) {
			System.out.println(e);
		}
	}
	/**
	 * this method is for counting how many media there are, and after that
	 * it return the amount of media .
	 * there is an IOException if it couldnt find any media.
	 * @param path
	 * @return return the amount of media there are.
	 */

	public int getAmountOfMedia(String path) {
		String[] values = new String[0];
		int counter = 0;
		try {
			values = ResourceReader.getValuesOnLine(path, counter);
			while(values != null) {
				values = ResourceReader.getValuesOnLine(path, counter);
				counter += 1;
			}
		} catch (IOException e) {
			System.out.println(e);
		}
		return counter;
	}
	/**
	 * this method is just like the one that load all the user, but its for media.
	 * The difference is that is gets how many media there are in the source file.
	 * And here it divide the media into two part, namely dvd and book. it creates
	 * an object for each one.
	 * @param path
	 */

	public void loadMedia(String path) {
		String[] values = new String[0];

		int size = getAmountOfMedia(path);

		media = new Hashtable<String, Media>(size);

		try {
			for(int i = 0; i < size-1; i++) {
				values = ResourceReader.getValuesOnLine(path, i);
				if(values[0].equals("Dvd")) {
					String[] actors = new String[values.length-4];
					for(int j = 0; j < actors.length; j++) {
						actors[j] = values[j+4];
					}
					media.put(values[1], new Dvd(values[1], values[2],
Integer.parseInt(values[3]), actors));
				} else {
					media.put(values[1], new Book(values[1], values[3],
Integer.parseInt(values[4]), values[2]));
				}
			}
		} catch (IOException e) {
			System.out.println(e);
		}
	}
}
```

LibraryView

```
package library;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;

/**
 * Main view for the program
 * @author tom.leonardsson
 *
 */
public class LibraryView extends JPanel {
		private LibraryController controller = new LibraryController();

		private LogInController logInController = new LogInController(controller);
		private LogInView logInView = new LogInView(logInController);

		private BorrowController borrowController = new BorrowController(controller, logInController);
		private BorrowView borrowView = new BorrowView(borrowController);

		private BorrowedController borrowedController = new BorrowedController(controller, logInController);
		private BorrowedView borrowedView = new BorrowedView(borrowedController);

		private JTabbedPane loggedInTabs = new JTabbedPane();

		private JButton accept = new JButton("Fortästt");

		/**
		 * Create the view and add all components and other views with a specifc controller
		 * @param controller the specfic controller to control the program with
		 */
		public LibraryView(LibraryController controller) {
				controller.setLogInController(logInController);

				loggedInTabs.add("LÅNA", borrowView);
				loggedInTabs.add("LÄMNA TILLBAMA", borrowedView);
				add(loggedInTabs);
				ButtonListener l = new ButtonListener();
				accept.addActionListener(l);
				logInView.add(accept, BorderLayout.EAST);

				this.controller = controller;

				this.add(logInView);
				loggedInTabs.setVisible(false);

				loggedInTabs.addChangeListener(new ChangeListener() {
						/**
						 * Listen for when the user swithces tabs and update the JTextArea that shows borrowed media
						 * @param arg0
						 */
						public void stateChanged(ChangeEvent arg0) {
								borrowedView.setTextToBorrowedList();
						}
		    });
		}

		/**
		 * Button listner
		 * @author tom.leonardsson
		 *
		 */
		private class ButtonListener implements ActionListener {
				/**
				 * Check for when the user presses accept after logging in to hide the log in view and show the logged in view
				 * @param e
				 */
				public void actionPerformed(ActionEvent e) {
						if(e.getSource() == accept) {
								if(controller.getLoggedInUser() != null) {
										logInView.setVisible(false);
										loggedInTabs.setVisible(true);
								}
						}
				}
		}
}
```

LogInController

```
package library;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;

/**
 * Main view for the program
 * @author tom.leonardsson
 *
 */
public class LibraryView extends JPanel {
			private LibraryController controller = new LibraryController();

			private LogInController logInController = new LogInController(controller);
			private LogInView logInView = new LogInView(logInController);

			private BorrowController borrowController = new BorrowController(controller, logInController);
			private BorrowView borrowView = new BorrowView(borrowController);

			private BorrowedController borrowedController = new BorrowedController(controller, logInController);
			private BorrowedView borrowedView = new BorrowedView(borrowedController);

			private JTabbedPane loggedInTabs = new JTabbedPane();

			private JButton accept = new JButton("Fortästt");

			/**
			 * Create the view and add all components and other views with a specifc controller
			 * @param controller the specfic controller to control the program with
			 */
			public LibraryView(LibraryController controller) {
						controller.setLogInController(logInController);

						loggedInTabs.add("LÅNA", borrowView);
						loggedInTabs.add("LÄMNA TILLBAMA", borrowedView);
						add(loggedInTabs);
						ButtonListener l = new ButtonListener();
						accept.addActionListener(l);
						logInView.add(accept, BorderLayout.EAST);

						this.controller = controller;

						this.add(logInView);
						loggedInTabs.setVisible(false);

						loggedInTabs.addChangeListener(new ChangeListener() {
									/**
									 * Listen for when the user swithces tabs and update the JTextArea that shows borrowed media
									 * @param arg0
									 */
									public void stateChanged(ChangeEvent arg0) {
												borrowedView.setTextToBorrowedList();
									}
			    });
			}

			/**
			 * Button listner
			 * @author tom.leonardsson
			 *
			 */
			private class ButtonListener implements ActionListener {
						/**
						 * Check for when the user presses accept after logging in to hide the log in view and show the logged in view
						 * @param e
						 */
						public void actionPerformed(ActionEvent e) {
									if(e.getSource() == accept) {
												if(controller.getLoggedInUser() != null) {
															logInView.setVisible(false);
															loggedInTabs.setVisible(true);
												}
									}
						}
			}
}
```

LogInView

```
package library;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

/**
 * View of the log-in screen
 * @author tom.leonardsson
 *
 */
public class LogInView extends JPanel {
		private LogInController controller;

		private JButton logIn = new JButton("LOGGA IN");

		private JPanel logInField = new JPanel();

		private JTextField logInId = new JTextField();

		private JLabel logInBoxText = new JLabel("ID: ");
		private JLabel loggedInUserProfile = new JLabel("PROFILE: ");

		/**
		 * Create a log-in view and add the components and panels with a specifc controller
		 * @param controller the specifc controller
		 */
		public LogInView(LogInController controller) {
				this.controller = controller;
				setLayout(new BorderLayout());
				logInField.setLayout(new GridLayout(0, 2));
				logInBoxText.setPreferredSize(new Dimension(80, logInBoxText.getSize().height));
				logInField.add(logInBoxText);
				logInField.add(logInId);
				this.add(logInField, BorderLayout.NORTH);
				this.add(loggedInUserProfile, BorderLayout.CENTER);
				this.add(logIn, BorderLayout.SOUTH);
				ButtonListener l = new ButtonListener();
				logIn.addActionListener(l);
		}

		/**
		 * ButtonListner
		 * @author tom.leonardsson
		 *
		 */
		private class ButtonListener implements ActionListener {
				/**
				 * Check for when the user presses log-in to check if the input is correct
				 * @param e
				 */
				public void actionPerformed(ActionEvent e) {
						if(e.getSource() == logIn) {
								controller.attemptLogIn(logInId.getText());
								loggedInUserProfile.setText(controller.getLoggedInProfileText());
						}
				}
		}
}
```

Program

```
package library;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

/**
 * View of the log-in screen
 * @author tom.leonardsson
 *
 */
public class LogInView extends JPanel {
				private LogInController controller;

				private JButton logIn = new JButton("LOGGA IN");

				private JPanel logInField = new JPanel();

				private JTextField logInId = new JTextField();

				private JLabel logInBoxText = new JLabel("ID: ");
				private JLabel loggedInUserProfile = new JLabel("PROFILE: ");

				/**
				 * Create a log-in view and add the components and panels with a specifc controller
				 * @param controller the specifc controller
				 */
				public LogInView(LogInController controller) {
								this.controller = controller;
								setLayout(new BorderLayout());
								logInField.setLayout(new GridLayout(0, 2));
								logInBoxText.setPreferredSize(new Dimension(80, logInBoxText.getSize().height));
								logInField.add(logInBoxText);
								logInField.add(logInId);
								this.add(logInField, BorderLayout.NORTH);
								this.add(loggedInUserProfile, BorderLayout.CENTER);
								this.add(logIn, BorderLayout.SOUTH);
								ButtonListener l = new ButtonListener();
								logIn.addActionListener(l);
				}

				/**
				 * ButtonListner
				 * @author tom.leonardsson
				 *
				 */
				private class ButtonListener implements ActionListener {
								/**
								 * Check for when the user presses log-in to check if the input is correct
								 * @param e
								 */
								public void actionPerformed(ActionEvent e) {
												if(e.getSource() == logIn) {
																controller.attemptLogIn(logInId.getText());
																loggedInUserProfile.setText(controller.getLoggedInProfileText());
												}
								}
				}
}
```