

```

package nthx.util.struts.navigation;

import org.apache.log4j.Logger;

import java.util.*;

import nthx.util.log.Log4jMain;

/** Object holds a short list of data which is a part of some bigger list.
 * It is used to maintain navigation through pages with many founded elements.
 * <br/>It is also used when navigating each book separately.
 *
 * @see nthx.util.struts.navigation.tests.NavigableDataTest
 * @see NavigableForm
 * @see NextPageAction ( with description how to use it in <tt>struts-config.xml</tt>
 * @see PrevPageAction
 * @see CustomPageAction
 *
 * @version $Id: NavigableData.java,v 1.2 2003/08/18 22:58:03 nthx Exp $
 * @author nthx@irc.pl
 */

public class NavigableData
{
    //--- Fields -----
    protected final static Logger logMain = Logger.getLogger(Log4jMain.class);

    public static final int DEFAULT_ITEMS_PER_PAGE = 10;
    //how many 'page links' to view on HTML page
    public static final int MAX_PAGES = 10;

    //collection with all elements to navigate through
    private Collection navigableElements;

    //list with items to view
    private List navigableList;

    //pages (numbers) user can view directly by clicking page's number
    private Map pages; //key=human number, value=number (starting from 0)

    //number of page, actually viewed
    //starts from 0
    private int actualPage;

    //number of pages which can be generated
    private int pagesNumber;

    //number of all items in a collection viewed
    private int allItemsNumber;

    //how many items can be viewed on a page
    public int itemsPerPage = DEFAULT_ITEMS_PER_PAGE;

    //--- Constructors -----
    public NavigableData(Collection collection)
    {
        this(collection, DEFAULT_ITEMS_PER_PAGE);
    }

    public NavigableData(Collection collection, int itemsPerPage)
    {
        setNavigableElements(collection);
        setItemsPerPage(itemsPerPage);
        updateNavigableList();

        setActualPage(0);
        updateNavigationVariables();
    }

    //--- Implementation -----
    public void goForward()
    {
        setActualPage(getNextPage());

        updateNavigationVariables();
        this.navigableList = getFromTheMiddle(getActualPage(), getItemsPerPage());
    }
}

```

```

public void goBackward()
{
    ...
}

public void goCustomPage(int customPage)
{
    ...
}

private void updateNavigationVariables()
{
    ...
};

private List getFromTheMiddle(int pageNumber, int itemsPerPage)
{
    List list = new ArrayList();
    int counter = 0;

    int from = pageNumber * itemsPerPage;
    int to = from + itemsPerPage - 1;

    logMain.debug("Getting elements from: " + from + " to: " + to);

    for (Iterator elements = getNavigableElements().iterator(); elements.hasNext(); )
    {
        Object element = elements.next();
        if (counter > to)
            break; //finished

        if (counter >= from)
        {
            list.add(element);
        }

        counter++;
    };
    return list;
};

/** Previous viewed page, counting from 0 */
public int getPrevPage(){
    if (getActualPage() > 0)
        return getActualPage() - 1;
    else
        return 0;
}

/** Next viewed page, counting from 0 */
public int getNextPage(){
    if (getActualPage() < getLastPage())
        return getActualPage() + 1;
    else
        return getActualPage();
}

/** Last viewed page, counting from 0 */
public int getLastPage()
{
    if (getPagesNumber() == 0)
        return 0;
    else
        return getPagesNumber() - 1;
}

/** Setting new data to the list also resets all counters, and variables
 * responsible for navigation.
 * Simple said: it resets all data.
 */
public void updateNavigableList(){
    updateNavigationVariables();
    setActualPage(0);

    this.navigableList = getFromTheMiddle(getActualPage(), getItemsPerPage());
}

```

```

//--- Getters and Setters ---
public void setActualPage(int actualPage)
{
    if (actualPage >= 0 && actualPage <= (getPagesNumber()-1))
        this.actualPage = actualPage;
    //when navigableElements.size() == 0
    else if (0 == actualPage && 0 == getPagesNumber())
        this.actualPage = actualPage;
    else
        throw new IllegalArgumentException(
            "To small/big page number: " + actualPage +
            " pages: " + getPagesNumber() +
            " navigableElements: " + getNavigableElements().size()
        );
}

...

public void setPages(Map pages) { this.pages = pages; }
public int getItemsPerPage(){ return itemsPerPage; }
public void setItemsPerPage(int itemsPerPage){ this.itemsPerPage = itemsPerPage; }
public int getAllItemsNumber() { return allItemsNumber; }
public void setAllItemsNumber(int allItemsNumber) { this.allItemsNumber = allItemsNumber; }

/** Actual viewed page, counting from 1. Human stands for 'humand readable'. */
public int getHumanActualPage(){
    return getActualPage() + 1;
};

/** Next viewed page, counting from 1. Human stands for 'humand readable'. */
public int getHumanNextPage(){
    return getNextPage() + 1;
};

/** Last viewed page, counting from 1. Human stands for 'humand readable'. */
public int getHumanLastPage(){
    return getLastPage() + 1;
};

/** Previous viewed page, counting from 1. Human stands for 'humand readable'. */
public int getHumanPrevPage(){
    return getPrevPage() + 1;
}

public Collection getNavigableElements(){ return navigableElements; }
public void setNavigableElements(Collection navigableElements){ this.navigableElements = navigableElements; }
}

```