Operating Systems Project – Page Replacement Algorithms

Goal: Create a page replacement simulation to compare and test algorithms.

Description: Section 8.4 in the textbook describes Page Replacement. Your program will implement FIFO and LRU page replacement. You will generate page-reference strings and supply them to each algorithm, counting the number of page faults in each and comparing the results. Run multiple tests, with different numbers of page frames and reference strings. Describe your results in your post-mortem.

Input: Read two args: the first is the length of the page references, the second is the # of frames

Output: Print out each of the page fault counts to System.out. (You may also wish to print out a visual description of the page references, similar to the textbook and our in-class practice).

Deliverables: Java program in an executable jar file named [YOUR_NAME]PageReplacement.jar, all source files (include your entire src folder retaining the folder structure), a readme.txt, and a post-mortem in a zip file named [YOUR_NAME]PageReplacement.zip.

All source code belongs in a package named edu.frostburg.cosc460.

[YOUR_NAME] = LastNameFirstName // e.g. KennedyStevePageReplacement.zip

Details:

Part 1: Basic framework

Begin by rereading section 8.4.

Create the class, PageGenerator, which generates page-reference strings with page numbers ranging from 0 to 9. The size of the reference string is passed to the constructor. Implement a getReferenceString() method that then returns a reference string as an array of integers.

Create a Driver class. It accepts two arguments from the command line. (The args[] array in main). See http://stackoverflow.com/questions/9168759/netbeans-how-to-set-command-line-arguments-in-java for tips on how to set that up in Netbeans. Right now, all your driver should do is read the arguments and get a reference string. Print it out to System.out so that you can see what you are generating.

Tip: Real page references are not purely random, so it will help test your algorithms if you don't make it completely random. Additionally, you may want to implement a way to test specific reference strings (specifically the ones from the slides and textbook) to make sure that your algorithms are functioning.

Part 2: Algorithms

Implement the following abstract class provided by our textbook authors:

```
public abstract class ReplacementAlgorithm
{
     // the number of page faults
     protected int pageFaultCount;
     // the number of physical page frame
     protected int pageFrameCount;
```

```
    // pageFrameCount - the number of physical page frames
    public ReplacementAlgorithm(int pageFrameCount)
    {
        if (pageFrameCount < 0)
            throw new IllegalArgumentException();
        this.pageFrameCount = pageFrameCount;
        pageFaultCount = 0;
    }

    // return - the number of page faults that occurred.
    public int getPageFaultCount()
    {
        return pageFaultCount;
    }

    // int pageNumber - the page number to be inserted
    public abstract void insert(int pageNumber);

}
```

You will create two concrete classes that extend ReplacementAlgorithm: one for LRU and one for FIFO. Each of these will need to implement the insert() method. Begin with FIFO and use the textbook's descriptions to guide you.

Test your implementation.

Next, implement LRU using a stack. (You may use Java's data structures for your stack. See: https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html).

Test your implementation.

Part 3: Wrap-up

Your driver class should now run each of your algorithms and compare their output. Describe the results of your algorithms in your post-mortem.

Make sure that you have well written and well documented code. Make sure that you have all deliverable items before you submit.

Test your program.

Bonus: Optimal

Implement a third algorithm: OPT. OPT is an unrealistic algorithm in that you get to use knowledge of future page references (down the string) unlike a real replacement algorithm. However, having OPT available allows you to make better comparisons with FIFO and LRU. The implementation details are up to you.

Checklist

	Item	Value
	Part 1	20%
	Part 2 • FIFO • LRU	50% (25%) (25%)
	Part 3	5%
	Bonus	+10%
	Code Documentation	10%
	Deliverables	5%
	Post-Mortem	10%

Avoidlist

	Item	Value
	Bugs	-x
	Runtime errors	-y
	Unsatisfactory quality	-z
	Compilation errors	-50+%