```java
package a1;
import java.util.ArrayList;
import java.util.HashMap;

public class OptimalPageReplacement {

    // Optimal Page Replacement Algorithm
    static int optimal(int referenceString[]) {
        // This array list will contain all the pages that are currently in memory
        ArrayList<Integer> pages = new ArrayList<Integer>();

        // This hashmap will store the future index of each page
        HashMap<Integer, Integer> indexes = new HashMap<>();

        int page_faults = 0, curPage, n = referenceString.length;

        for (int i = 0; i < n; i++) {
            curPage = referenceString[i];

            // Check if the set can hold more pages
            if (pages.size() < 3) {  // Assuming pageFrames = 3
                // Insert it into set if not already present
                if (!pages.contains(curPage)) {
                    pages.add(curPage);
                    // Increment page fault count
                    page_faults++;
                    displayPageFrames(pages, page_faults);
                }

                // Store the future index of the page
                indexes.put(curPage, findNextIndex(curPage, i, referenceString));
            }
            // If the set is full, need to select a page to be replaced
            else {
                // Check if current page is not already present in the set
                if (!pages.contains(curPage)) {
                    // Find a page that is referenced farthest in the future
                    int optimal = Integer.MIN_VALUE, pageToBeReplaced = 0;
                    int temp;
                    for (int j = 0; j < pages.size(); j++) {
                        temp = pages.get(j);
                        if (indexes.get(temp) > optimal) {
                            optimal = indexes.get(temp);
                            pageToBeReplaced = j;
                        }
                    }

                    // Remove the selected page from memory
                    indexes.remove(pages.get(pageToBeReplaced));
                    pages.set(pageToBeReplaced, curPage);
                    // Increment page faults
                    page_faults++;
                    displayPageFrames(pages, page_faults);
                }

                // Update the current page index
                indexes.put(curPage, findNextIndex(curPage, i, referenceString));
            }
        }
        return page_faults;
    }

    // Function to find the next index of a page
    static int findNextIndex(int curPage, int curIndex, int[] referenceString) {
        // Starting at the current index, find the index of future use of the page
        int i;
        for (i = curIndex + 1; i < referenceString.length; i++) {
            if (referenceString[i] == curPage) {
                break;
            }
        }
        return i;
    }

    // Function to display the current state of page frames
    static void displayPageFrames(ArrayList<Integer> pages, int page_faults) {
        System.out.print("At PageFault- " + page_faults + " :: Pages- ");
        for (int i = 0; i < pages.size(); i++) {
            System.out.print(" " + pages.get(i));
        }
        System.out.print("\n");
    }

    // Driver method to test the algorithm
    public static void main(String[] args) {
        int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1};

        // Number of frames in memory
        int pageFrames = 3;

        // Calling optimal page replacement
        System.out.println("--- Implementing Optimal Page Replacement Algorithm
-----");
        int pageFaults = optimal(pages);
        System.out.println("Number of page faults = " + pageFaults);
    }
}
```