

```

package LinearneStrukture;

/**
 * Implementacija Reda na statički način korišćenjem niza.
 * Red je linearna struktura koja funkcioniše po
 * FIFO (First In First Out) principu, odnosno prvi element
 * koji je ubačen u red se prvi izbacuje. Elementi se
 * ubacuju na jednom, a izbacuju sa drugog kraja strukture.
 *
 * @author Kostandin Dimitrijević, Strukture podataka i algoritmi, FON, 2010
 */
public class StatickiRed {
    // niz u kome se čuvaju elementi reda
    private int[] niz;
    // dimenzija niza - maksimalni broj elemenata u redu
    private int dimenzija;
    // indeks prvog elementa u redu (početak reda)
    private int p;
    // indeks poslednjeg elementa u redu (kraj reda)
    private int k;
    // broj elemenata u redu
    private int brElemenata;

    /**
     * Konstruktor. Inicijalizuje sve promenljive koje se koriste u implementaciji
     * @param kapacitet : maksimalni broj elemenata u redu
     */
    public StatickiRed(int kapacitet){
        dimenzija = kapacitet;
        niz = new int[dimenzija];
        brElemenata = 0;
        p = 0;
        k = -1;
    }

    /**
     * Proverava da li je red pun
     * @return true ako je red pun. U suprotnom false.
     */
    public boolean punRed(){
        return brElemenata == dimenzija;
    }

    /**
     * Proverava da li je red prazan.
     * @return true ako je red prazan. U suprotnom false.
     */
    public boolean prazanRed(){
        return brElemenata == 0;
    }

    /**
     * Ubacuje novi element u red
     * @param element vrednost koja se ubacuje u red
     * @return true ako je element ubačen, false ako nema mesta (red je pun)
     */
    public boolean enqueue(int element){
        if (punRed())
            return false;

        // prvi način
        k++;
        if (k == dimenzija)
            k = 0;
        niz[k] = element;
        // drugi način
        //niz[k = ++k % dimenzija] = element;
        brElemenata++;
        return true;
    }
}

```

```

/**
 * Izbacuje prvi element iz reda (element na početku)
 * @return izbačeni element sa početka reda.
 * @throws Exception Ako je red prazan, baca se izuzetak.
 */
public int dequeue() throws Exception{
    if (prazanRed())
        throw new Exception("Red je prazan");

    int podatak = niz[p];
    p++;
    if (p == dimenzija)
        p = 0;
    brElemenata--;
    return podatak;
}

/**
 * Vraća element koji se nalazi na početku reda
 * @return prvi element u redu.
 * @throws Exception Ako je red prazan, baca se izuzetak.
 */
public int pocetak() throws Exception{
    if (prazanRed())
        throw new Exception("Red je prazan");

    return niz[p];
}

/**
 * Vraća element koji se nalazi na kraju reda
 * @return poslednji element u redu.
 * @throws Exception Ako je red prazan, baca se izuzetak.
 */
public int kraj() throws Exception{
    if (prazanRed())
        throw new Exception("Red je prazan");

    return niz[k];
}

/**
 * Ispisuje sve elemente reda od početka do kraja (for petlja)
 */
public void ispisi(){
    if (prazanRed()) return;
    for (int i = p; i != k; ){
        System.out.println(niz[i]);
        // prvi način
        //i = ++i % dimenzija;

        // drugi način
        i++;
        if (i == dimenzija)
            i = 0;
    }
    System.out.println(niz[k]);
}

/**
 * Ispisuje sve elemente reda od početka do kraja (while petlja)
 */
public void ispisi2(){
    int i = p;
    int br = 0;
    while (br > brElemenata){
        System.out.println(niz[i]);
        // prvi način
        //i = ++i % dimenzija;

        // drugi način
        i++;
        if (i == dimenzija)
            i = 0;
    }
}
}

```