

```

package LinearneStrukture;

/**
 * Sadrži algoritme za sortiranje niza celih brojeva u rastućem redosledu.
 * Svi implementirani algoritmi imaju kompleksnost n kvadrat.
 *
 * @author Kostandin Dimitrijević, Strukture podataka i algoritmi, FON, 2010
 */
public class Sortiranje {

    /**
     * Sortiranje metodom selekcije - selection sort.
     * Sortiranje u rastućem redosledu se vrši tako što se prolazi kroz
     * nesortirani deo niza i pronalazi najmanji element i on se stavlja na
     * kraj sortiranog dela niza. Ovo se ponavlja dok se ne sortiraju svi elementi.
     *
     * @param niz Niz koji se sortira
     */
    public void selectionSort(int[] niz){
        // prolazimo od prvog do pretposlednjeg elementa, jer kada obiđemo
        // ove elemente, poslednji će biti sortitran
        for (int i = 0; i < niz.length - 1; i++) {
            // indeks najmanjeg elementa
            int min = i;
            // prolazimo kroz ostatak strukture i tražimo minimalni element
            for (int j = i + 1; j < niz.length; j++) {
                if (niz[j] < niz[min])
                    min = j;
            }
            // ako minimalni nije već na poziciji, zameni mesta minimalnom elemntu
            // i elementu na kraju sortiranog dela.
            if (min != i){
                int pom = niz[i];
                niz[i] = niz[min];
                niz[min] = pom;
            }
        }
    }

    /**
     * Sortiranje metodom ubacivanja - insertion sort.
     * Sortiranje u rastućem redosledu se vrši tako što se prolazi kroz
     * nesortirani deo niza i uzima prvi (trenutni) element i zatim se unazad
     * prolazi kroz sortirani deo i svi elementi koji su veći od
     * trenutnog elementa se pomeraju za jedno mesto unapred ostavljajući mesta
     * za premeštanje trenutnog elementa. Na kraju se trenutni element upisuje
     * u oslobođeno mesto. Ovo se ponavlja dok se ne sortiraju svi elementi.
     *
     * @param niz Niz koji se sortira.
     */
    public void insertionSort(int[] niz){
        for (int i = 1; i < niz.length; i++) {
            // trenutni element
            int pom = niz[i];
            int j = i;
            // prolazak unazad do početka strukture ili dok se ne nađe prvi
            // element koji je manji od trenutnog
            while(j > 0 && niz[j-1] > pom){
                // pomeranje elemenata jedno mesto unapred
                niz[j] = niz[j-1];
                j--;
            }
            // upisivanje trenutnog elementa na oslobođenu poziciju
            niz[j] = pom;
        }
    }
}

```

```

    }
}

/**
 * Mehurasti sort - bubble sort.
 * Sortiranje u rastućem redosledu se vrši tako što se prolazi kroz niz i
 * trenutni element se upoređuje sa svakim sledbenikom. Ako nisu u odgovarajućem
 * redosledu menjaju im se mesta. Ovo se ponavlja za svaki element.
 *
 * @param niz Niz koji se sortira.
 */
public void bubbleSort(int[] niz){
    for (int i = 0; i < niz.length - 1; i++) {
        // niz[i] je trenutni element
        for (int j = i + 1; j < niz.length; j++) {
            if (niz[i] > niz[j]){
                // zamena mesta elementima
                int pom = niz[i];
                niz[i] = niz[j];
                niz[j] = pom;
            }
        }
    }
}

/**
 * Shell sort.
 * Varijanta insertion sort algoritma. Niz se deli na logičke podnizove.
 * Elementi svakog logičkog podniza su udaljeni za inc mesta.
 * Na ove logičke podnizove se primenjuje insertion sort. Nakon toga se
 * inc duplo smanjuje i ponavlja algoritam dok je inc veće od nule.
 *
 * @param niz Niz koji se sortira.
 */
public void shellSort(int[] niz){
    // radi što boljih performansi inc bi trebalo da se kreće u rasponu
    // od 3 do 8
    int inc = 3;
    while (inc > 0){
        // primena insertion sorta na podnizovima
        for (int i = 1; i < niz.length; i++) {
            // trenutni element
            int pom = niz[i];
            int j = i;
            // upoređuju se trenutni element sa svojim logičkim prethodnikom
            // koji se nalazi na inc mesta od njega
            while(j >= inc && niz[j-inc] > pom){
                // pomeranje unapred
                niz[j] = niz[j-inc];
                j -= inc;
            }
            niz[j] = pom;
        }
        inc = inc/2;
    }
}
}

```