Compito di astrazione

Testo

Prendere il seguente esempio di codice Java e modificarlo in modo da realizzarne uno che abbia (almeno) un metodo in grado di ordinare o in maniera crescente o in maniera decrescente i dati.

Tale metodo deve poter essere invocato indicando, in qualche modo, la direzione di ordinamento.

1. specificare il metodo (ed eventuali metodi accessori)
2. implementarlo

(Suggerimento: usare qualsiasi tipo di astrazione sembri essere adatto; anche più di uno.)

```
public class MyInsertionSort {

    static int[] arr1 = {10,34,2,56,7,67,88,42};
    static int temp;

    public static void main(String a[]){
        for (int i = 1; i < arr1.length; i++) {
            for(int j = i ; j > 0 ; j--){
                if(arr1[j] < arr1[j-1]){
                    temp = arr1[j];
                    arr1[j] = arr1[j-1];
                    arr1[j-1] = temp;
                }
            }
        }
        for(int i:arr1){
            System.out.print(i);
            System.out.print(", ");
        }
    }
}
```

Correzioni

Nel seguito fornisco alcuni esempi di commenti di vari tipi di soluzione. Cercando di evidenziare le cose positive e quelle negative.

Versione con 2 metodi quasi uguali

```
...
    main (...){ ...
        if(crescente==true){
         InsertionSortCrescente(arr1);
        }else if (crescente==false){
         InsertionSortDecrescente(arr1);
        }

public static void InsertionSortCrescente(int[] a){
       for (int i = 1; i < arr1.length; i++) {
              for(int j = i ; j > 0 ; j--){
                  if(arr1[j] < arr1[j-1]){
                      temp = arr1[j];
                      arr1[j] = arr1[j-1];
                      arr1[j-1] = temp;
                  }
              }
          }
    }

public static void InsertionSortDecrescente(int[] a){
      for (int i = 1; i < arr1.length; i++) {
             for(int j = i ; j > 0 ; j--){
                 if(arr1[j] > arr1[j-1]){
                     temp = arr1[j];
                     arr1[j] = arr1[j-1];
                     arr1[j-1] = temp;
                 }
             }
         }
```

Miei commenti

- soluzione molto povera: ridondanza di codice dei due metodi che ordinano
- brutta: `InsertionSortCrescente` dichiara un parametro a e invece usa arr1
- senza riuso: manca un metodo `swap` che invece verrebbe usato in due parti
- senza commenti che descrivono le astrazioni/i contratti

Soluzione senza sotto-metodi

```
public static void sort(int[] arr1, int x) {
        // ordina in modo crescente se x=0
        if (x == 0) {
            for (int i = 1; i < arr1.length; i++) {
                for (int j = i; j > 0; j--) {
                    if (arr1[j] < arr1[j - 1]) {
                        temp = arr1[j];
                        arr1[j] = arr1[j - 1];
                        arr1[j - 1] = temp;
                    }
                }
            }
            for (int i : arr1) {
                System.out.print(i);
                System.out.print(", ");
            }
        } else if (x == 1) {
            // ordina in modo decrescente se x=1
            for (int i = 1; i < arr1.length; i++) {
                for (int j = i; j > 0; j--) {
                    if (arr1[j] > arr1[j - 1]) {
                        temp = arr1[j];
                        arr1[j] = arr1[j - 1];
                        arr1[j - 1] = temp;
                    }
                }
            }
```

Miei commenti

- mancano i due metodi per i due rami dell'if
- e comunque i due rami sono quasi uguali tra loro: no riuso, no astrazioni
- manca un metodo swap
- senza commenti che descrivono le astrazioni/i contratti

Soluzione intermedia

```
public static void main(String a[]) throws IOException{

        int[] array = {130,6,59,81,728,142,32,62,94,553,1662,1,43,5,522};
        InputStreamReader input = new InputStreamReader(System.in);
        char choice;

        System.out.println();
        System.out.println("Benvenuto! Ho in memoria un vettore. In che direzione lo vuoi or

        System.out.println("Digita 'C' per l'ordine crescente oppure digita 'D' per l'ordine
        choice = (char)input.read();

        chooseSort (array, choice);

        printArray(array);
    } // end main

        /**
        * chooseSort int[] a; It allows you to choose the sort direction
        * @param a an array of integers, REQUIRE to have 1 or more elements
        * @param c is a char variable, REQUIRE to be a valid character.
        **/
        private static void chooseSort (int [] a, char c){

            if(c=='C'){
                ascendingSort(a);
                }

                else if (c=='D'){
                    descendingSort(a);
                    }

                    else {
                        System.out.print("Errore nell'input immesso. Ricontrolla ciÃ² che ha
                        System.exit(0);
                    }

        }


        /**
          * ascendingSort int[] a; MODIFY the array a by sort ascending
          * @param vector an array of integers, REQUIRE to have 1 or more elements
```

```
  * @param i an index of the array, REQUIRE to be a valid index and < vector.length
  * @param j an index of the array, REQUIRE to be a valid index and > 0.
**/
private static void ascendingSort (int[] vector){
    for (int i = 1; i < vector.length; i++) {
       for(int j = i ; j > 0 ; j--){
            if(vector[j] < vector[j-1]){
               swap (vector, j);
           }
       }}
}



/**
  * descendingSort int[] a; MODIFY the array a by sort descending
  * @param vector an array of integers, REQUIRE to have 1 or more elements
  * @param i an index of the array, REQUIRE to be a valid index and < vector.length
  * @param j an index of the array, REQUIRE to be a valid index and > 0.
**/
private static void descendingSort (int[] vector){
    for (int i = 1; i < vector.length; i++) {
           for(int j = i ; j > 0 ; j--){
                if(vector[j] > vector[j-1]){
                   swap (vector, j);
               }
           }}
    }


/**
* swap a[k] with a[k-1]; MODIFY the array a
* @param a an array of integers, REQUIRE to have 2 or more elements
* @param k an index of the array, REQUIRE to be a valid index and > 0.
**/
private static void swap (int[] a, int k) {
    int temp;
    temp = a[k];
    a[k] = a[k-1];
    a[k-1] = temp;
}
```

Miei commenti

- soluzione buona che ha i contratti
- bene per astrazione procedurale e per parametrizzazione
- non bene l'aver definito due metodi quasi uguali

Soluzione senza specifiche

```
public class MyInsertionSort {
     static int[] arr1 = {10,34,2,56,7,67,88,42};
        //static int temp;
     Arrays.sort(arr1);

     for(int i:arr1){
         System.out.print(i);
         System.out.print(", ");
     }

       /* public static void main(String a[]){
            for (int i = 1; i < arr1.length; i++) {
                for(int j = i ; j > 0 ; j--){
                    swap(j);
                }
            }
            stampa();
        }

        private static void stampa() {
            for(int i:arr1){
                System.out.print(i);
                System.out.print(", ");
            }
        }

        private static void swap(int j) {
            if(arr1[j] < arr1[j-1]){
                temp = arr1[j];
                arr1[j] = arr1[j-1];
                arr1[j-1] = temp;
            }
        }*/
    }
```

Miei commenti

- pessima soluzione: non ci sono commenti che descrivono l'astrazioni
- la soluzione non ordina l'array

Altra brutta soluzione

```
public class Ord {
    static int[] arr1 = {10,34,2,56,7,67,88,42};
    static int temp;
    static int direzione = 1; // con direzione=0 l'ordinamento è decrescente con direzione=
        public static void main (String a[], int direzione){
       if (direzione == 1){
          for (int i = 1; i < arr1.length; i++) {
              for(int j = i ; j > 0 ; j--){
                  if(arr1[j] < arr1[j-1]){
                      temp = arr1[j];
                      arr1[j] = arr1[j-1];
                      arr1[j-1] = temp;
                  }
              }
          }
       }else{
         if(direzione == 0){
             for (int i = 1; i < arr1.length; i++) {
                   for(int j = i ; j > 0 ; j--){
                       if(arr1[j] > arr1[j-1]){
                           temp = arr1[j];
                           arr1[j] = arr1[j-1];
                           arr1[j-1] = temp;
                       }
                   }
              }
         }
       }
```

Miei commenti

- non ci sono sottometodi
- non ci sono commenti che descrivono contratti
- codice duplicato

Soluzione con espressione condizionale

la sua soluzione:

```java
/**
 * This class permits to order a given array by invoking the method doInsertionSort(...);
 */
public class MyInsertionSort {

    private static final Boolean ASCENDING = true;
    private static final Boolean DESCENDING = false;
    private static int[] arr;


    /**
     * Take in input an array, set it and invoke the method doInsertionSort(Boolean mode);
     * @param array is an array of integer, REQUIRE to be not null;
     * @param mode is a boolean variable that determinate the order; use the static constan
     */
    public static void doInsertionSort(int[] array, Boolean mode) {
        setArray(array);
        doInsertionSort(mode);
    }

    /**
     * Use the set arr and MODIFY his order in relation of the input mode (ascending or des
     * @param mode is a boolean variable that determinate the order; use the static constan
     */
    /**
     * @param mode
     */
    public static void doInsertionSort(Boolean mode) {
        if (arr.length > 1) {
            for (int i = 1; i < arr.length; i++) {
                for (int j = i ; j > 0 ; j--) {
                    if ((mode) ? (arr[j] < arr[j-1]) : (arr[j] > arr[i])) { // Contracted f
                        swap(arr, j, j-1);
                    }
                }
            }
        }
    }

    /**
     * Swap the position of two elements; MODIFY array;
```

```
     * @param array of integer, REQUIRE to have 2 or more elements;
     * @param i index of array;
     * @param j index of array;
     */
    public static void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }


    /**
     * set the array input in the static variable; MODIFY arr;
     * @param array is an array of integer;
     */
    public static void setArray(int[] array) {
        arr = array;
    }


    /**
     * show at terminal the static variable arr; as the EFFECT to show arr;
     */
    public static void toMonitor() {
        System.out.print(arr[0]);
        for (int i = 1; i < arr.length; i++) {
            System.out.print(", ");
            System.out.print(arr[i]);
        }
    }

    /**
     * @return the static constant ASCENDING;
     */
    public static boolean getAscending() {
        return ASCENDING;
    }

    /**
     * @return the static constant DESCENDING;
     */
    public static boolean getDescending() {
        return DESCENDING;
    }

    /**
```

```
     * main method for TESTING;
     */
    public static void main(String[] args) {
        int[] array = {10,34,2,56,7,67,88,42};
        doInsertionSort(array, getAscending());
        toMonitor();
    }

}
```

Miei commenti

- bene per le astrazioni e la definizione dei contratti
- bene per le definizioni delle costanti
- non bene l'uso della variabile globale `arr`
- bene per l'idea di usare x?s1:s2,

```
if ((mode) ? (arr[j] < arr[j-1]) : (arr[j] > arr[i])) {
   swap(...);
}
```

- ma che risulta in codice poco leggibile e error-prone;
- meglio sarebbe stato scrivere

```
if (mode && (arr[j]<arr[j-1])){
      swap(arr,j, j-1)
} else if (!mode && arr[j]>arr[i]) {
      swap(arr,j,j-1)
}
```

Soluzione “offensiva” nei miei confronti

```
public class insert {

    static int[] arr1 = {10,34,2,56,7,67,88,42};
    static int temp;

    public static void main(String[] args){
        for (int i = 1; i < arr1.length; i++) {
            for(int j = i ; j > 0 ; j--){
                if(op (arr1[j],arr1[j-1],args[0])){
                    temp = arr1[j];
                    arr1[j] = arr1[j-1];
                    arr1[j-1] = temp;
                }
            }
        }
        for(int i:arr1){
            System.out.print(i);
            System.out.print(", ");}

     }

    // il seguente codice confronta i parametri e in base al metodo di ordinamento fa esegu

    public static boolean op(int j,int j1,String arg){
        if ((arg.equals("crescente")&& (j<j1) )||(arg.equals("decrescente")&& (j>j1) ))
        return true;
        else return false;

    }
    }
```

Miei commenti

- pessima, non corretta
- nemmeno provato ad eseguirla! (manca lo swap)
- nessun contratto
- nessuna parametrizzazione
- nessuna astrazione

Soluzione con specifche non adeguate

```
public class Ordninamento {
    static int vett[], l, n;

    /*Questo metodo crea l'array da ordinare chiedendo al utente di inserire il
     * numero di elementi da ordinare(tale numero in seguito viene usato per definire la lu
     * del vettore), in seguito viene chiesto al utente di inserire gli
     * elementi che vuole ordinare.
     *
     */
    private static void CreaVettDaOrdinare(){

        int m, i = 0;
        Scanner buff = new Scanner(System.in);
        System.out.println("Inserisci il numero di elementi da ordinare");
        l = buff.nextInt();
        vett = new int [l];
        System.out.println("Inserisci i numeri da ordinare");

        while(i < l){
            m = buff.nextInt();
            vett[i] = m;
            i++;
        }
    }
    /*
     * Questo metodo chiede al utente se vuole ordinare
     * in numeri in modo crescente o decrescente
     */
    private static void SceltaDelOrdinamento(){

        Scanner input = new Scanner(System.in);
        System.out.println("Inserisci 0 per avere un ordinamento crescente oppure 1 per ave
        n = input.nextInt();


        if (n == 0){
            Crescente(vett);
        }else{
            Decrescente(vett);
        }

    }
```

```
/*
 * Ordina i numeri in modo crescente.
 * il metodo mette a confronto l'elemento in posizioni i con
 * l'elemento in posizione i+1 e gli scambia di posizione se
 * Ã¨ neccessario
 */

private static void Crescente(int vett []){
     int temp;
    for (int i = 1; i < vett.length; i++) {
            for(int j = i ; j > 0 ; j--){
                if(vett[j] < vett[j-1]){
                    temp = vett[j];
                    vett[j] = vett[j-1];
                    vett[j-1] = temp;
                }
            }
    }

}
/*
 * Ordina i numeri in modo decrescente.
 * il metodo mette a confronto l'elemento in posizioni i con
 * l'elemento in posizione i+1 e gli scambia di posizione se
 * Ã¨ neccessario
 */

private static void Decrescente(int vett[]){
    int temp;
     for (int i = 1; i < vett.length; i++) {
            for(int j = i ; j > 0 ; j--){
                if(vett[j] > vett[j-1]){
                    temp = vett[j];
                    vett[j] = vett[j-1];
                    vett[j-1] = temp;
                }
            }
     }
}



public static void main(String args[]){
    CreaVettDaOrdinare();

    SceltaDelOrdinamento();
```

```
        for(int i:vett){
                System.out.print(i);
                System.out.print(", ");
            }
    }
}
```

Miei commenti

- bene per i metodi con i parametri
- manca lo swap, che verrebbe riusato 2x
- ingenua la ripetizione dei due metodi `crescente` e `decrescente`

Soluzione senza specifiche

```
public class MyInsertionSort{
    static int[] arr1 = {10,34,2,56,7,67,88,42} ;
    static int temp ;

    public static void main(String[] args){

        // variables
        Scanner in = new Scanner(System.in) ;
        boolean flag = false ;
        String choice = "_" ;
        boolean ascending = true ;

        // choice order
        do{
            System.out.print("type\n a , for an ascending sort,\n d , for a decreasing sort.
            choice = in.next() ;
            if(choice.equals("a")){
                flag = true ;
            }
            if(choice.equals("d")){
                flag = true ;
                ascending = false ;
            }
        }while(!flag) ;

        // execution order
        if(ascending){
            ascendingSort() ;
        }else{
            decreasingSort() ;
        }

        // print sorted array
        for(int i:arr1){
                    System.out.print(i);
                    System.out.print(", ");
            }
            System.out.print("\n");

    }    // end main

    private static void ascendingSort(){
        for (int i=1;i<arr1.length;i++){
```

```
            for(int j=i;j>0;j--){
                if(arr1[j]<arr1[j-1]){
                    temp = arr1[j] ;
                    arr1[j] = arr1[j-1];
                    arr1[j-1] = temp;
                        }
            }
        }
    }

    private static void decreasingSort(){
        for (int i=1;i<arr1.length;i++){
            for(int j=i;j>0;j--){
                if(arr1[j]>arr1[j-1]){
                    temp = arr1[j] ;
                    arr1[j] = arr1[j-1];
                    arr1[j-1] = temp;
                           }
            }
        }
    }

}    // end class
```

Miei commenti

- male per l'assenza dei commenti che danno la specifica
- male per l'assenza di parametri
- male per assenza di metodo swap riusato
- male per duplicazione di codice tra i due metodi di ordinamento
- a che serve dire "//end class"?

Altre soluzioni possibili

Le seguenti due soluzioni sono "orientate agli oggetti", la prima possibile in java 7 e 8, la seconda solo con java 8.

Uso di comparator generici

```
package it.uniud.poo.abstractions;

/**
 * @author giorgio
 * Example with parametric choice of sort direction.
 *
 */
public class MySortAnonymousClasses {

    /**
     * Labels to specify order direction
     */
    public  enum sortDirection {
        INCREASING,
        DECREASING
    }

    /**
     * Interface to be used as a parameter to
     * implement a particular kind of comparison
     * between int.
     */
    interface IntComparator {
        /**
         * @param x
         * @param y
         * @return true or false depending on what meaning we want to
         * give to compare. Eg. compare(x,y) can imply x<y, or x=2*y, or ...
         * In the context of sorting only x<y or x>y are useful choices.
         */
        boolean compare(int x, int y);
    }
```

```
/**
 * Run a simple example with two sorting procedures: up and down.
 */
public static void main (String a[]){
int[] arr1 = {10,34,2,56,7,67,88,42};

doParametricSort(arr1, sortDirection.INCREASING);
System.out.format("Increasing: ");
for(int i:arr1){
    System.out.print(i);
    System.out.print(", ");
}
doParametricSort(arr1, sortDirection.DECREASING);
System.out.format("\nDecreasing: ");
for(int i:arr1){
    System.out.print(i);
    System.out.print(", ");
    }
}
/**
 * sort the array a
 * MODIFY the array a so that values are ordered
 * @param a: an array of integers to be sorted
 * @param dir: the direction of the sort: INCREASING/DECREASING
 */
/**
 * @param a
 */
private static void doParametricSort(int[] a, sortDirection dir) {
    IntComparator ic = null; // the actual comparator that we will be using
    switch (dir) {
    case INCREASING:
        ic = new IntComparator() {
            @Override
            public boolean compare(int x, int y) {
                return (x < y);
            }
        };
        break;

    case DECREASING:
        ic = new IntComparator() {
                @Override
                public boolean compare(int x, int y) {
                    return (x > y);
                }
```

```
            };
            break;
    }

    doInsertionSort(a, ic);
}
/**
 * Sort the array a using the direction implied by the
 * generic comparator. MODIFY array a.
 * @param a
 * @param ic: a generic comparator for integers.
 */
private static void doInsertionSort(int[] a, IntComparator ic) {
    for (int i = 1; i < a.length; i++) {
        for (int j = i ; j > 0 ; j--){
            //if (a[j] < a[j-1]){
            if (ic.compare(a[j],a[j-1])){
                swap(a, j);
            }
        }
    }
}
/**
 * swap a[j] with a[j-1]; MODIFY the array a
 * @param a an array of integers, REQUIRED to have 2 or more elements
 * @param j an index of the array, REQUIRED to be a valid index and > 0.
 */
private static void swap(int[] a, int j) {
    int temp;
    temp = a[j];
    a[j] = a[j-1];
    a[j-1] = temp;
}
}
```

Commenti

- si noti la definizione di una enumerazione per indicare la direzione; meglio che non una stringa o un intero o un booleano. Più esplicito, parlante!
- Si noti la definizione di una interfaccia; consente di disaccoppiare chi usa da chi definisce.
- si noti la creazione di due IntComparator con *classi anonime*;
- e infine il modo di fare il confronto;
- **Importante**: `doInsertionSort` non sa nulla su come si confrontano gli elementi dell'array; il deciderlo e saperlo è compito di `doParametricSort`.

Che a sua volta delega alle due classi anonime il *come fare* il confronto.

Uso di lambda

```
package it.uniud.poo.abstractions;

import java.util.function.BiPredicate;

/**
 * @author giorgio
 * Example with parametric choice of sort direction implemented with lambdas.
 *
 */
public class MySortLambda {

    /**
     * Labels to specify order direction
     */
    public  enum sortDirection {
        INCREASING,
        DECREASING
    }


    /**
     * Run a simple example with two sorting procedures: up and down.
     */
    public static void main (String a[]){
    int[] arr1 = {10,34,2,56,7,67,88,42};

    doParametricSort(arr1, sortDirection.INCREASING);
    System.out.format("Increasing: ");
    for(int i:arr1){
        System.out.print(i);
        System.out.print(", ");
    }
    doParametricSort(arr1, sortDirection.DECREASING);
    System.out.format("\nDecreasing: ");
    for(int i:arr1){
        System.out.print(i);
        System.out.print(", ");
        }
    }
    /**
     * sort the array a
```

```
     * MODIFY the array a so that values are ordered
     * @param a: an array of integers to be sorted
     * @param dir: the direction of the sort: INCREASING/DECREASING
     */
    private static void doParametricSort(int[] a, sortDirection dir) {
        BiPredicate<Integer, Integer> ic = null; // we need to use Integer
        switch (dir) {
        case INCREASING:
            ic = (Integer x, Integer y) -> (x < y);// first lambda
            break;

        case DECREASING:
            ic = (Integer x, Integer y) -> (x > y);// second lambda
                break;
        }

        doInsertionSort(a, ic);
    }
    /**
     * Do sort the array a using the direction implied by the
     * generic predicate. MODIFY array a.
     * @param a
     * @param ic: a lambda predicate of two Integers.
     */
    private static void doInsertionSort(int[] a, BiPredicate<Integer, Integer> ic) {
        for (int i = 1; i < a.length; i++) {
            for (int j = i ; j > 0 ; j--){
                if (ic.test(a[j],a[j-1])){// automatic type conversion
                                          // from int to Integer
                    swap(a, j);
                }
            }
        }
    }
    /**
     * swap a[j] with a[j-1]; MODIFY the array a
     * @param a an array of integers, REQUIRED to have 2 or more elements
     * @param j an index of the array, REQUIRED to be a valid index and > 0.
     */
    private static void swap(int[] a, int j) {
        int temp;
        temp = a[j];
        a[j] = a[j-1];
        a[j-1] = temp;
    }
}
```

Commenti

- si noti che rispetto a prima cambia solo il metodo `doParametricSort` e la segnatura di `doInsertionSort`;
- si noti che rispetto a prima, il programma è più conciso;
- anche qui `doInsertionSort` non sa come fare il confronto; è stato delegato a `doParametricSort` che stavolta non delega il *come fare* il confronto ad altri oggetti.