Corso sul linguaggio Java

Modulo JAVA7

A1– Vector, Arrays e StringBuffer

Prerequisiti

- Programmazione base in Java
- Utilizzo di classi e oggetti
- Algoritmi notevoli sul vettore

Introduzione

In questa Unità vediamo in pratica i vantaggi dell'allocazione dinamica della memoria, tramite le seguenti classi Java:

- **Vector**
- **Arrays**
- **StringBuffer**

Allocazione dinamica

La tecnica dell'allocazione dinamica in Java è molto frequente, in quanto ottimizza l'utilizzo della memoria e, grazie al **garbage collector**, *non richiede al programmatore la conseguente deallocazione.*

In generale, la sintassi per allocare dinamicamente un oggetto è:

classe ident = **new** *costruttore()*;

dove:

- *classe* è il nome di una classe
- *ident* è il nome dell'oggetto allocato
- *costruttore()* è il costruttore della classe (con o senza parametri)

Allocazione dinamica

```
public class AllocazioneDinamica
{
    public static void main(String args[ ])
    {
        String s = new String ();
        Integer n = new Integer (3);
        s="Numero";
        System.out.println (s + " " + n);
    }   /* fine main */
}   /* fine classe */
```

Classe **String**

Classe **Integer**

La classe *Vector*

La classe **Vector** implementa in Java un **array dinamico di oggetti** che ha funzionalità e metodi simili agli array in quanto:

- la selezione degli elementi si effettua mediante un *indice*;
- richiede il *tipo* degli elementi, come per gli array:
  - **Vector <Integer>** v = **new Vector <Integer>** ();
  - **Vector <String>** v = **new Vector <String>** (10);

Tuttavia, un oggetto **Vector** ha le ulteriori caratteristiche:

- viene *ridimensionato dinamicamente* (*run-time*) ed è quindi più flessibile rispetto agli array;
- richiede il package **java.util.Vector**; (oppure **java.util.*;**)
- può *ospitare elementi di tipo eterogeneo* (classi diverse, v. seguito)

I metodi della classe *Vector*

METODO	EFFETTO
Vector <*tipo*> *ident* =**new Vector** <*tipo*>()	Crea un vettore vuoto di nome *ident*
Vector <*tipo*> *ident* =**new Vector** (**int** *n*)	Crea un vettore *ident* vuoto di *n* elementi
Vector<*tipo*> *ident* =**new Vector (int** *n*, **int** *inc*)	Crea un vettore *ident* di n elementi ed aggiunge *inc* elementi ogni volta che viene riempito.
void addElement(**Object** *obj*)	Aggiunge l'elemento *obj* in coda e incrementa di uno la dimensione del vettore
void add (**int** *index*, **Object** *obj*)	Aggiunge l'elemento *obj* in posizione *index*
void clear()	Rimuove tutti gli elementi dal vettore
boolean contains (**Object** *obj*)	Test di esistenza dell'elemento *obj* nel vettore
Object elementAt (**int** *index*)	Restituisce l'oggetto presente in posizione *index*
Object setElementAt (**Object** *obj*, **int** *index*)	Sostituisce *obj* all'oggetto della posizione *index*
int indexOf (**Object** *obj*)	Restituisce la prima posizione dell'elemento *obj*
Object firstElement ()	Restituisce l'oggetto presente in prima posizione
Object lastElement ()	Restituisce l'oggetto presente in ultima posizione
boolean isEmpty()	Test vettore vuoto
void insertelementAt (**Object** obj, **int** *index*)	Inserisce elemento *obj* in posizione *index*
void removeElementAt (**int** *index*)	Rimuove elemento in posizione *index*
int size()	Restituisce il numero di componenti presenti
String toString ()	Converte in stringa ciascun elemento del vettore

Lettura e stampa

ATTIVITA': scrivere un'applicazione che legga da input 5 nomi e li aggiunga via via in un **Vector** *v*. Successivamente, stampare *v*.

```
import java.util.*;
import java.io.*;
public class LeggiStampa
{   public static void main(String[] args)  throws Exception
    {   InputStreamReader In = new InputStreamReader (System.in);
        BufferedReader tastiera = new BufferedReader (In);
        Vector <String> v = new Vector <String>(1,1);
        for (int i=0; i<5; i++)
            v.addElement(tastiera.readLine());
        System.out.println( v );
        v.clear();
        System.out.println( v );
    }
}
```

Aggiunge in coda gli elementi letti

Stampa *v*

Azzera tutto il contenuto

Inserimento e azzeramento

ATTIVITA': scrivere un'applicazione che carichi in un **Vector** *v* di interi i valori 152, 17 e 25, stampi il vettore *v* e il numero di elementi contenuti. Successivamente, azzeri *v* e effettui un test di vettore vuoto.

```
import java.util.*;
public class Vari
{   public static void main(String[] args)
    {  Vector <Integer> v= new Vector <Integer>(1,1);
       System.out.println( v.size() );
       v.add(0, 152);
       v.add(1, 17);
       v.add(2, 25);
       System.out.println (v);
       System.out.println(v.size() );
       v.clear();
       System.out.println(v.isEmpty() );
    }
}
```

Aggiunge in posizione specificata

Stampa la dimensione

Test vettore vuoto

Indice e posizione

ATTIVITA': scrivere un'applicazione che carichi in un **Vector** *v* i nomi Anna, Elio, Gianni, Elena e successivamente ricerchi la posizione di Elio e stampi primo e ultimo elemento.

```
import java.util.*;
public class Indexes
{
   public static void main(String[] args)
   {
         Vector <String>v = new Vector <String> ();
         v.add("Anna");
         v.add("Elio");
         v.add("Gianni");
         v.add("Elena");
         System.out.println( v.indexOf("Elio") );
         System.out.println( v.firstElement());
         System.out.println( v.lastElement());
}
```

Ricerca posizione

Primo elemento

Ultimo elemento

Vector con dati eterogenei

Come possiamo sfruttare la classe **Vector** per memorizzare dati di classi diverse?

È sufficiente creare una classe **Vector** di **Object** grazie al fatto che **Object** è superclasse di tutte le classi.

```
// Dichiarazione Vector
Vector <Object> v = new Vector <Object> ();
// Creazione dati
Dati d = new Dati();                // crea oggetto di classe Dati
Persona p = new Persona();          // crea oggetto di classe Persona
// Inserimento oggetti nel Vector
v.addElement(d);  v.addElement(p)   // aggiunge in fondo al Vector v;
```


Vector con dati eterogenei

Ma poi come estraiamo gli elementi dal **Vector** se hanno tutti subito il *cast implicito* verso la classe **Object**?

```
d = v.elementAt (0);
p = v.elementAt (1);
```

Errore *run-time* poiché il compilatore non può sapere a quale classe appartenga l'oggetto estratto

Usando il *cast esplicito* si riportano gli elementi del **Vector** alla loro classe di appartenenza.

```
d = (Dati) v.elementAt (0);
p = (Persona) v.elementAt (1);
```

Vector con dati eterogenei

```
import java.lang.*;
import java.util.*;
public class Casting
 { public static void main(String args [])
   {  Vector <Object>a = new Vector <Object>(10);
      Integer x = new Integer (5);
      Double y = new Double (3.5);
      String z = new String ("cielo");
      a.add(x);     a.add(y);     a.add(z);
      System.out.println("a: " + a);
      int intx = ((Integer) a.elementAt(0)).intValue();
      double doubley = ((Double) a.elementAt(1)).doubleValue();
      String stringz = (String) a.elementAt(2);
      System.out.println (intx);
      System.out.println (doubley);
      System.out.println (stringz);
   }
}
```

Crea 3 oggetti di classe diversa

Esegue il *cast esplicito* per estrarre gli elementi

Tipo array e classe Vector

Principali differenze sintattiche tra array e **Vector**

1. **int** v[] = **new int** [10],	**Vector <Integer>** v=**new Vector<Integer>** (10);
2. Non richiede package	**java.util.Vector;**
3. **for** (**int** i=0 **;** i<v.**length** ; i++) ...	**for** (**int** i=0 **;** i<v.**size**() ; i++) ...
1. v[i] = 5	v.**setElementAt**(5, i);
2. **System.out.println** (v[3]);	**System.out.println** (v.**elementAt**(3));
3. Per la stampa serve un ciclo	**System.out.println** (v);
4. **for** (**int** i=4; i<v.**length**; i++) v[i-1]=v[i]; v[v.**length**-1]=**null**;	v.**removeElementAt** (3);
8. **for** (**int** i=v.**length**-1;i >2 ; i--) v[i] = v[i-1]; v[2] = 8;	**v.insertElementAt** (8, 2);

La classe **Arrays**

Questa classe contiene i metodi per operare con gli **algoritmi notevoli sugli array** (ricerca e ordinamento) e per svolgere altre operazioni.

Nella tabella, *tipo* indica **Integer**, **Double**, **Object**,… e deve essere lo stesso nell'ambito di ciascun metodo.

METODO	EFFETTO
static List asList (**Object** *v[]*)	Restituisce gli elementi dell'array *v[]* in una lista
static int binarySearch (*tipo t[ ], tipo k*)	Cerca nell'array *t[ ]* il valore *k*
static boolean equals (*tipo t1[ ], tipo t2[ ]*)	Testa se i due array sono uguali
static void fill (*tipo t[ ], tipo v*)	Assegna a ciascuna componente di *t[ ]* il valore *v*
static void sort (*tipo t[ ]*)	Ordina l'array *t[ ]*
static void arraycopy (*tipo a1[], int offset1, tipo a2[], int offset2, int numEl);*	Copia *numEl* elementi di *a1*, partendo da *offset1*, in *a2*, partendo da *offset2*

La classe **Arrays**

ATTIVITA': Dato un elenco di nomi, stamparlo in ordine alfabetico.

```
import java.util.*;
public class Sorting
{   public static void main(String[] args)
    {   String nomi[] = {"Saverio", "Anna", "Lucio"};
        Arrays.sort (nomi);
        for (int i=0; i<3; i++)
        {
            System.out.print (nomi[i] + " ");
        }
    }
}
```

Un modo più pratico per stampare l'array è:
System.out.println (**Arrays.asList** (nomi));

La classe **StringBuffer**

La classe **StringBuffer** definisce stringhe che possono essere modificate dinamicamente nel tempo, a differenza degli oggetti di classe **String** *che sono manipolabili ma non modificabili*.

Gli oggetti di classe **StringBuffer**:

- possono essere modificabili come *contenuto* e *dimensione*
- possono ospitare stringhe di lunghezza variabile
- sono convenienti rispetto a quelli di classe **String** poiché evitano la creazione di nuove stringhe come risultati intermedi di manipolazioni.

I metodi di **StringBuffer**

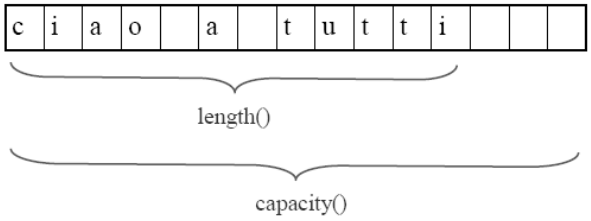
La classe **StringBuffer** offre la possibilità di modificare una stringa (gli oggetti di classe **String** possono essere manipolati ma non modificati)

METODO	EFFETTO
StringBuffer ()	Crea uno stringbuffer vuoto
StringBuffer (**int** *length*)	Crea uno stringbuffer di lunghezza *length*
StringBuffer(**String** *str*)	Crea uno stringbuffer e lo inizializza con *str*
StringBuffer append (*tipo t*)	Converte *t* in stringa e l'aggiunge allo stringbuffer
int length()	Dà il numero di caratteri dello stringbuffer
int capacity()	Dà il numero di caratteri complessivamente allocati
char charAt (**int** *index*)	Dà il carattere in posizione *index* dello stringbuffer
StringBuffer deleteCharAt (**int** *index*)	Elimina il carattere in posizione *index*
StringBuffer delete (**int** *start*, **int** *end*)	Elimina i caratteri da posizione *start* a *end*
int indexOf (**String** *str*)	Dà la posizione di *str* all'interno dello stringbuffer
StringBuffer insert (**int** index, *tipo t*)	Inserisce in posizione *index* di stringbuffer il dato *t*
StringBuffer replace (**int** *start*, **int** *end*, **String** *str*)	Sostituisce con *str* una sottostringa di stringbuffer
void setCharAt (**int** *index*, **char** *ch*)	Pone *ch* in posizione *index* dello stringbuffer
String substring (**int** *start*)	Dà la sottostringa di stringbuffer a partire da *start*
String substring (**int** *start*, **int** *end*)	Dà la sottostringa di stringbuffer tra *start* e *end*
String toString ()	Converte lo stringbuffer in **String**

lenght() e capacity()

In figura è indicato il significato dei due metodi

- **int lenght()**, che dà la lunghezza della stringa
- **int capacity()**, che dà lo spazio complessivo allocato



Esempi di **StringBuffer**

```
public class OperazioniStringBuffer
{   public static void main(String args[ ])
    {  StringBuffer buf = new StringBuffer("The Java Class Libraries");
       System.out.println (buf.substring(4, 8));          // "Java"
       System.out.println (buf.substring(9));// "Class Libraries"
       buf.replace(4, 8, "C++");
       System.out.println (buf.toString());    // "The C++ Class Libraries"
       buf.delete(3, 7);
       System.out.println (buf.toString());    // "The Class Libraries"
       buf.deleteCharAt(0); buf.deleteCharAt(0); buf.deleteCharAt(0);
       buf.insert(0, "new Java");
       System.out.println (buf.toString());    // "new Java Class Libraries"
       buf.setCharAt(0, 'N');
       System.out.println (buf.toString());    // "New Java Class Libraries"
    }       /* fine main */
}   /* fine classe */
```

Argomenti

- Allocazione dinamica
- La classe **Vector**
- I metodi della classe **Vector**
- Lettura e stampa
- Inserimento e azzeramento
- Indice e posizione
- **Vector** di **Object**
- La classe **Arrays**
- La classe **StringBuffer**
- I metodi di **StringBuffer**