ARRAY IN JAVA

Link ai video di questa lezione:

Video 1: https://www.youtube.com/watch?v=j1Fy5O2UI6k&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=27

Video 2: https://www.youtube.com/watch?v=E1-nmE-UQ6o&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=28

Per dichiarare un array (monodimensionale o bidimensionale) si usa la seguente sintassi:

```
tipo [] nomeArray              // oppure    tipo nomeArray[]
tipo[][] nomeArray             // per le matrici
```

L'array in Java però, diversamente dal C, è esso stesso un oggetto, quindi anche se viene dichiarata, non è ancora presente nello heap finchè non ne viene invocato i costruttore con il token "new".

Per questo motivo non è necessario indicare in fase di dichiarazione il numero di elementi dell'array con una costante (come accadeva in C). **Quando l'Array viene effettivamente istanziato con "new", invece, va specificato il numero di elementi di cui è composto. Questo e' importantissimo! Quando si usa il new per istanziare l'array, a quel punto va specificata la dimensione dell'array! La differenza con il C è che tale dimensione può essere determinata a runtime e quindi, ad esempio, acquisita con input da tastiera!**

Ad esempio:

```
int[] arrayInteri;
arrayInteri = new int[5];      //crea un array di interi.
                               //Ogni cella contiene il valore 0 (valore di default)
```

Naturalmente le due precedenti espressioni possono essere unite in una sola

```
int[] arrayInteri = new int[5];
```

E' possibile istanziare un array senza la parola chiave "new" solo quando si inizializza direttamente l'array con l'elenco dei valori contenuti, ad esempio:

```
int[] arrayInteri={32,23,44,32,5};
```

Per accedere ai singoli elementi dell'array si usa la sintassi già nota con l'indicazione dell'elemento fra parentesi quadre, ad esempio:

```
arrayInteri[3]=122;  (all'elemento in posizione 3 dell'array viene assegnato il valore 122)
```

Essendo un oggetto, l'array possiede un attributo (pubblico) che ne indica la dimensione, tale attributo è "**length**" ed è comodo da utilizzare nei cicli for che prevedono un contatore da 0 fino alla dimensione dell'array.

```
for (i=0;i<arrayInteri.length;i++)
        …….
```

Esercizi (fare da soli):

1. Costruire con un ciclo, nel main, un array monodimensionale di interi contenente 10 valori inseriti da tastiera. Visualizzare, con un altro ciclo for il contenuto dell'array.

2. Costruire nel main con un ciclo nidificato una matrice (array bidimensionale) contenente la tavola pitagorica (dall' 1 al 10). Visualizzare con un altro ciclo nidificato la tavola pitagorica creata

Array di oggetti

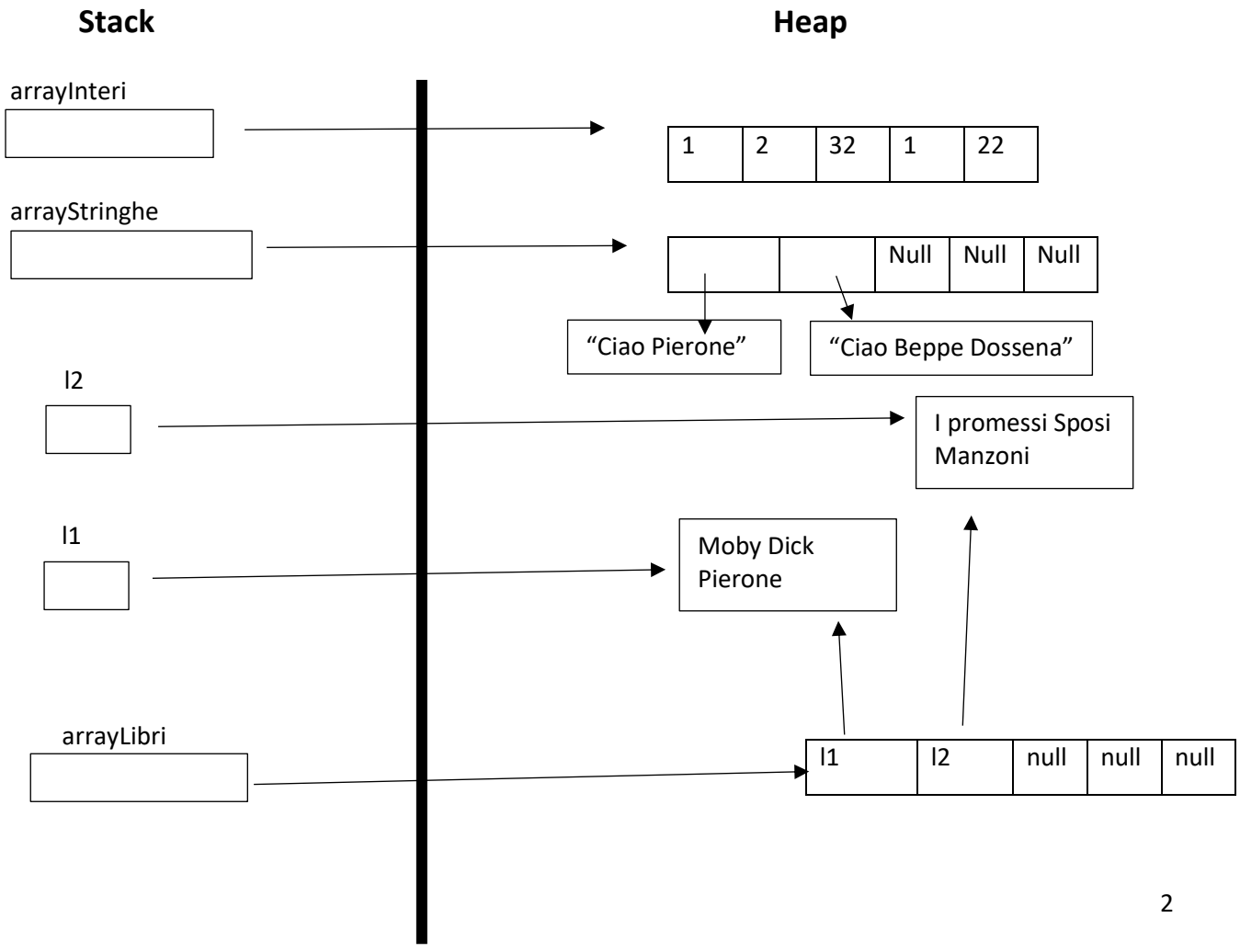
Fin' ora abbiamo parlato di array contenenti tipi di dati interi. Un array può contenere due tipologie di dati, e ciò comporta alcune differenze.

Gli elementi dell'array possono essere:

a. Tipo di dati nativi in questo caso nelle "celle" dell'array nello heap vengono memorizzati direttamente i valori dei dati.

b. Reference ad oggetti: quando il dato memorizzato nell'array è un oggetto, nelle celle dell'array non viene memorizzato l'oggetto ma un reference (riferimento) all'oggetto.

Graficamente le due situazioni si possono mostrare nel seguente modo:



Quando gli elementi di un array di interi non vengono inizializzati, tali elementi assumono valore di default 0

Quando gli elementi di un array di oggetti non vengono inizializzati, tali elementi sono dei reference a **null** (nessun oggetto).

Per **eliminare** un elemento dall'array di oggetti , il contenuto dell'array in quella posizione viene posto a null. Ad esempio:

arrayLibri[1]=null

Quando nel codice si cerca di accedere ad un elemento dell'array in una posizione superiore alla dimensione si genera un errore in fase di esecuzione (run time), anzi, il nome corretto dell'evento è "**eccezione**". L'eccezione di "sforamento" di un array è la seguente:

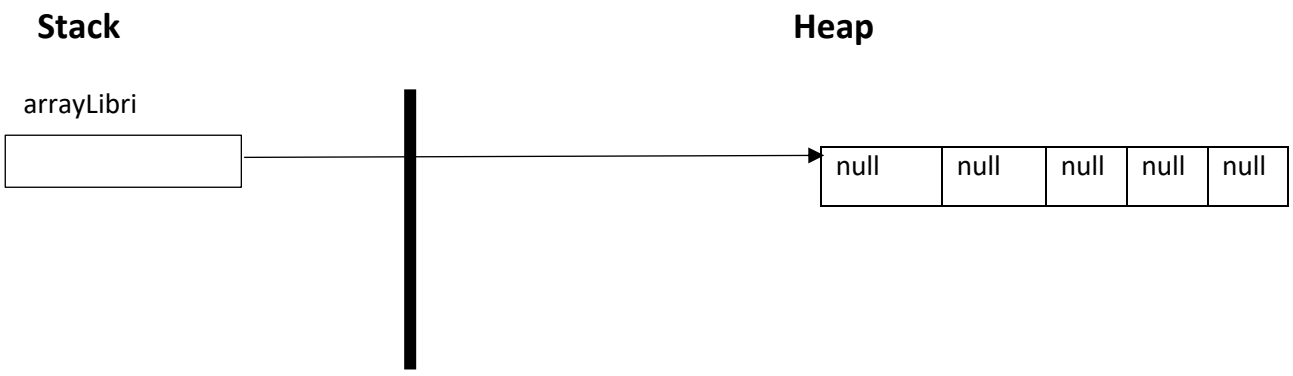
`ArrayIndexOutOfBoundsException`

Impareremo in seguito a gestire diversi tipi di eccezioni, per ora è importante riconoscerle per individuare eventuali errori in fase di debug. Se incontreremo un'eccezione sapremo che il problema è certamente nell'aver "sforato" l'accesso ad un array durante l'esecuzione.

Quando si aggiunge un elemento ad un array è sempre necessario porre attenzione a ciò che accade nello heap. In particolare bisogna prestare attenzione al modo con cui viene aggiunto l'elemento, perché questo può portare conseguenze molto diverse nella gestione degli oggetti. Verranno ora mostrati due modi diversi per assegnare dei libri ad un array di libri.

A. **Copiare il reference di un oggetto già esistente nella cella dell'array (non va bene poichè c'è dipendenza fra gli oggetti)**

```
Libro [] arrayLibri = new Libro[5]   //   creazione array con 5 "celle" che possono ospitare
                                     //   reference ad oggetti di tipo libro. Inizialmente le celle
                                     //   sono vuote (contengono puntatori a null)
```



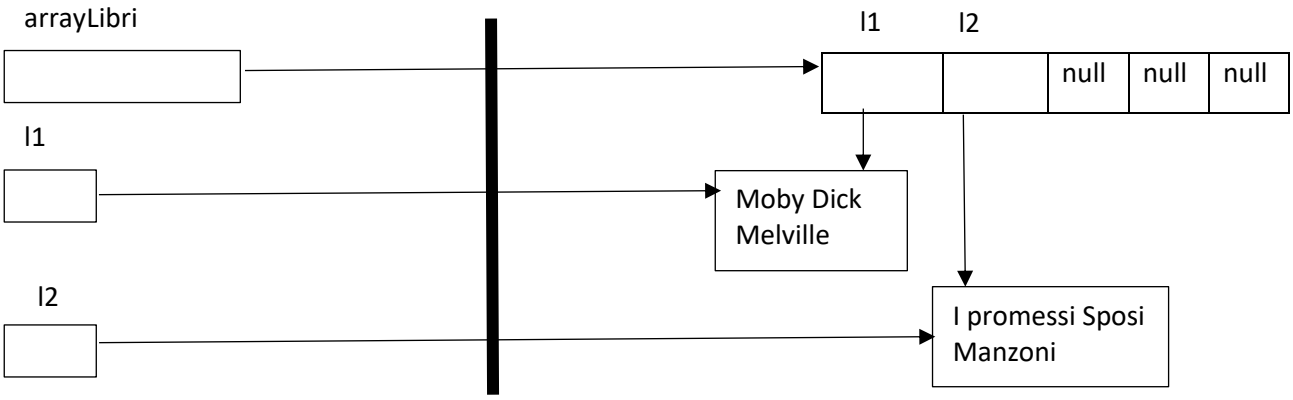
Se ora vengono istanziati due oggetti di tipo libro (l1 e l2) e poi tali libri vengono assegnati all'array, la situazione è la seguente:

Libro l1 = new Libro ("Moby Dick", "Melville", 600)

Libro l2= new Libro ("I promessi sposi", "Manzoni", 500)

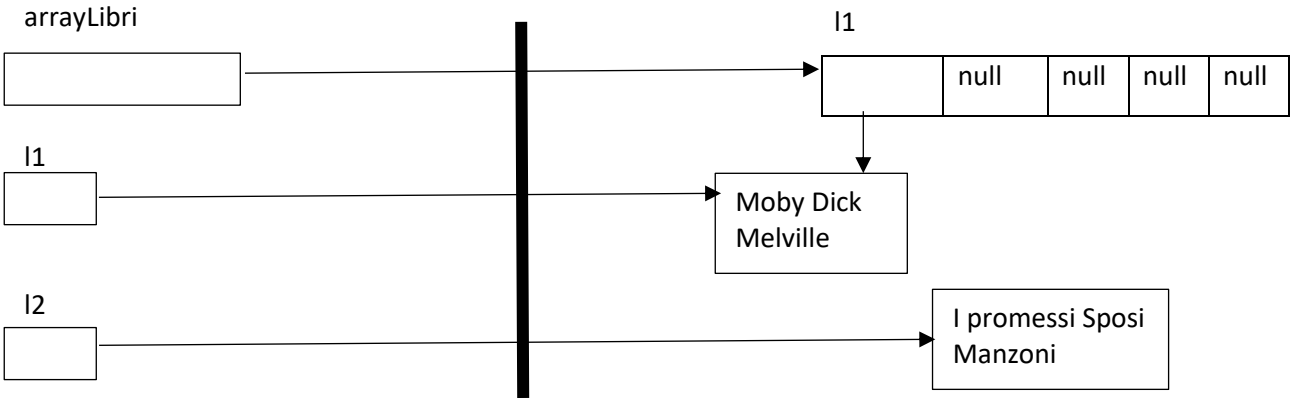
arrayLibri [0]=l1;

arrayLibri[1]=l2;



Se l'elemento in posizione 1 dell'array viene posto a null, comunque l'oggetto l2 rimane nello heap poiché è referenziato (puntato) dal reference l2:

arrayLibri[1]=null



OSSERVAZIONE: Si osservi che se viene modificato il libro l1, ovviamente anche il libro contenuto nell'array (che è sempre l1) viene modificato, quindi si ha dipendenza fra l'oggetto "arrayLibri" e l'oggetto "l1", cosa che generalmente non deve accadere! Non deve accadere poiché questa dipendenza fa sì che modificando un oggetto (l1), viene modificato anche un altro oggetto (arrayLibri)!

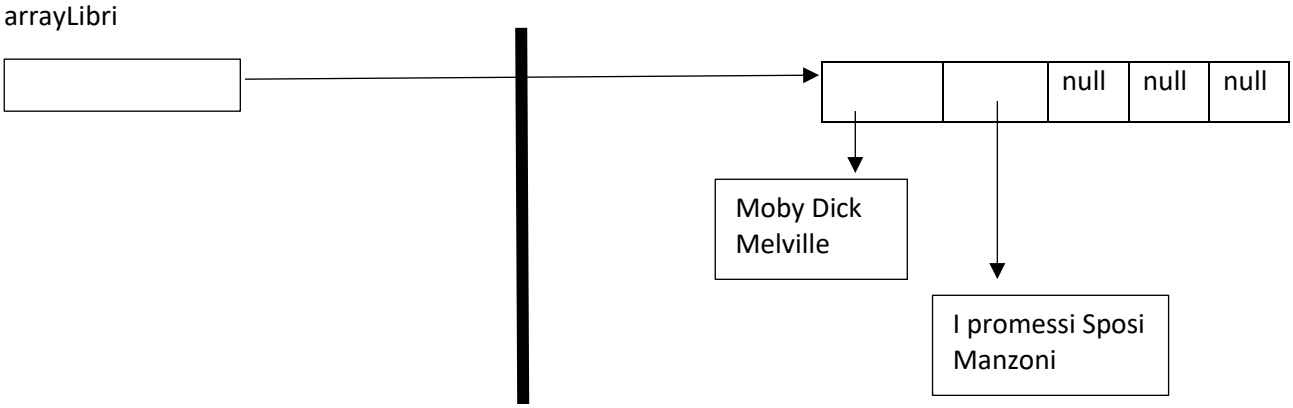
B. Costruire un NUOVO oggetto dell'array con il costruttore

Se invece gli oggetti libro vengono aggiunti direttamente nell'array invocando il costruttore la situazione è la seguente:

```
Libro [] arrayLibri = new Libro[5]   //   creazione array con 5 "celle" che possono ospitare
                                     //   reference ad oggetti di tipo libro. Inizialmente le celle
                                     //   sono vuote (contengono puntatori a null)
```

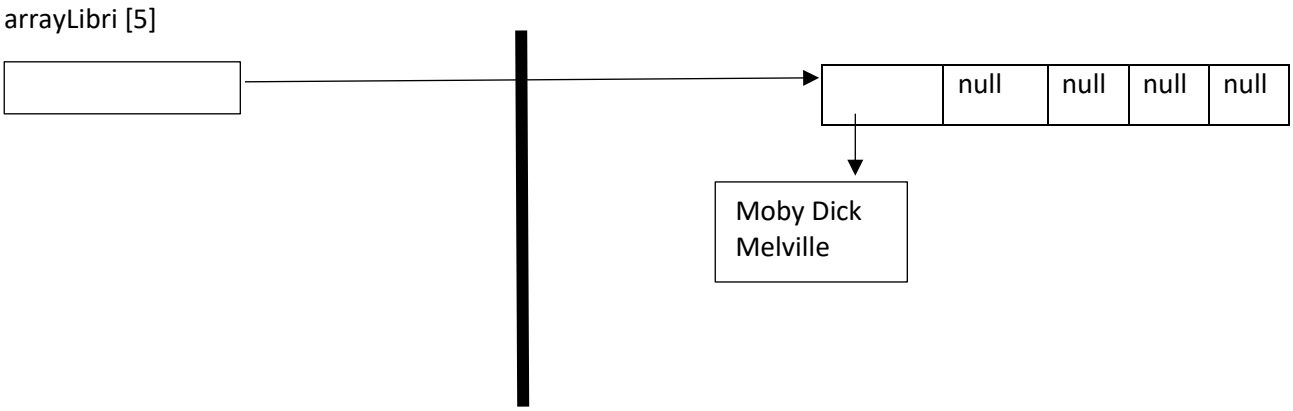
arrayLibri[0]= new Libro ("Moby Dick", "Melville", 600)

arrayLibri[1]= new Libro ("I promessi sposi", "Manzoni", 500)



In questo caso se l'elemento in posizione 1 dell'array viene posto a null, l'oggetto da esso puntato (il libro "I promessi sposi") viene eliminato dal garbage collector perché non è più referenziato:

arrayLibri[1]=null



La modalità B si può realizzare anche ricorrendo al costruttore di copia della classe Libro. **In questo modo si garantisce l'indipendenza fra gli oggetti "arrayLibri" e i singoli libri passati come parametro**. Questo è il modo migliore. Il codice è il seguente:

Libro l1 = new Libro (“Moby Dick”, “Melville”, 600)

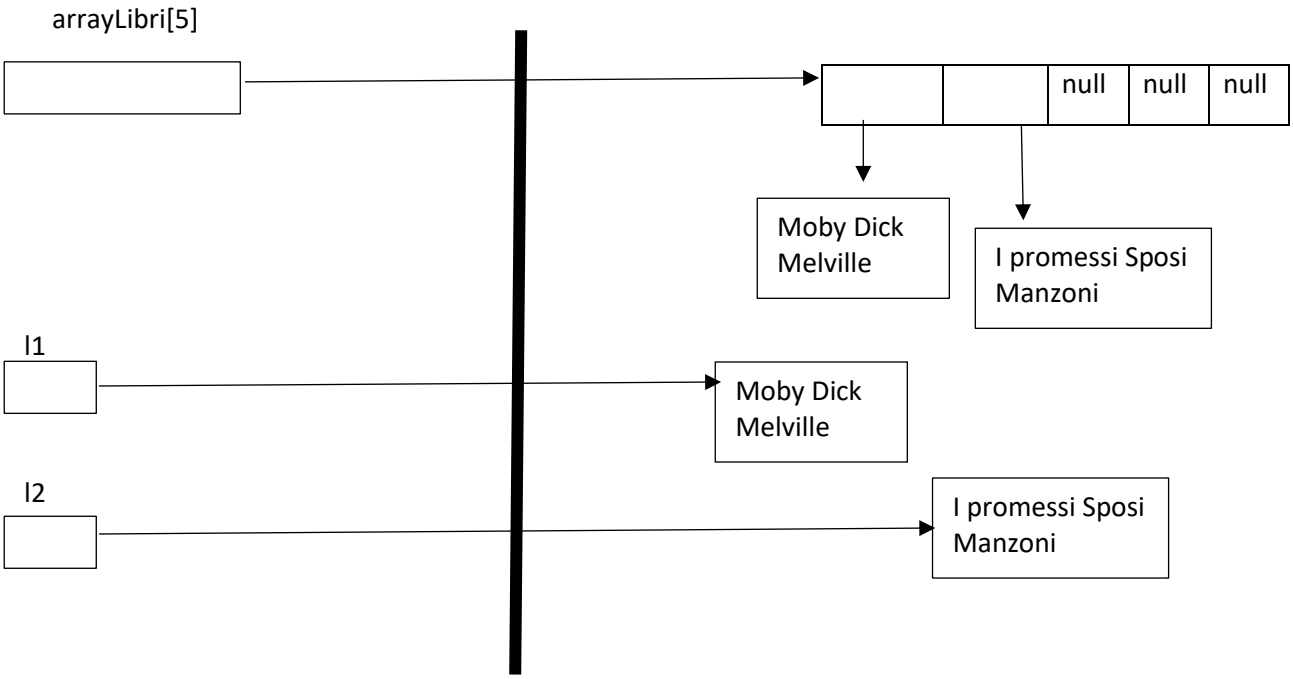
Libro l2= new Libro (“I promessi sposi”, “Manzoni”, 500)

arrayLibri [0]=**new** Libro(l1);

arrayLibri[1]=**new** Libro (l2);

ATTENZIONE: rispetto al caso A c’è la parola chiave new!!

La situazione che viene a crearsi nella memoria centrale è la seguente:



Il modo corretto per istanziare l’array di libri è dunque quello del caso A o quello del caso B?

Tranne in alcuni casi particolari, in cui il programmatore “vuole” mantenere la dipendenza fra gli oggetti, il principio generale della OOP di indipendenza degli oggetti fa sì che sia preferibile la modalità B.

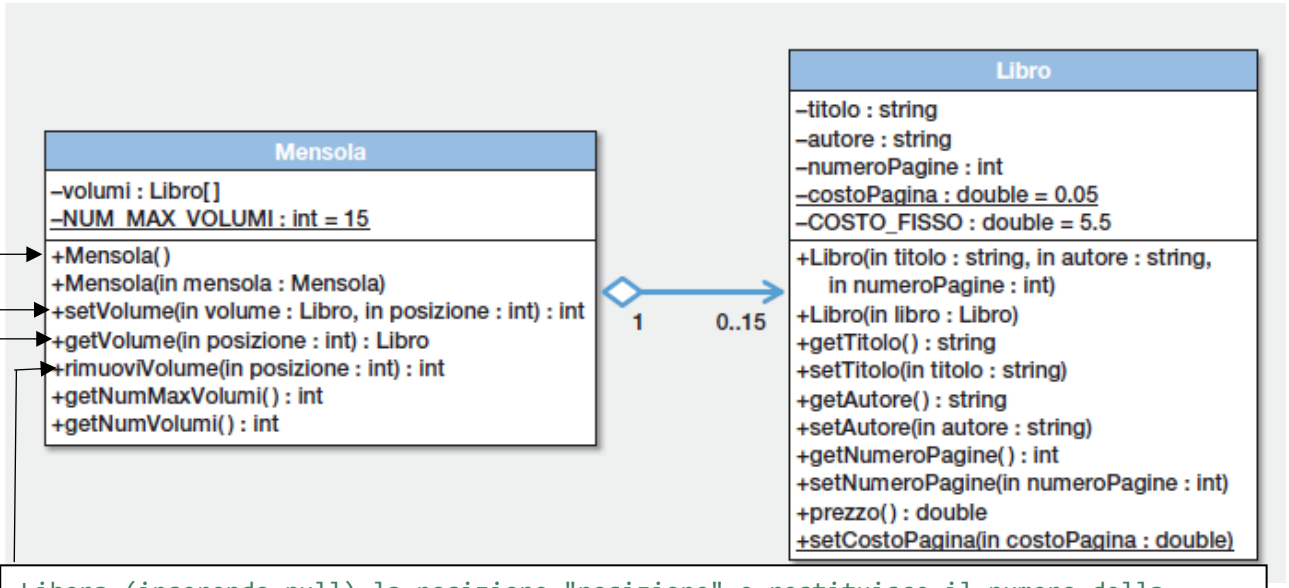
SI modifichi il progetto Libreria aggiungendo una classe Mensola in associazione di aggregazione con la classe Libro (codice disponibile più avanti)

Link al video:

Video1: https://www.youtube.com/watch?v=C8txyRPOSPU&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=29

Video2: https://www.youtube.com/watch?v=-F_NjqesSoU&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=30

```
Costruttore: istanzia l'attributo volumi (un array di libri)
"vuoto". Tale array conterrà 15 volte Null.
```

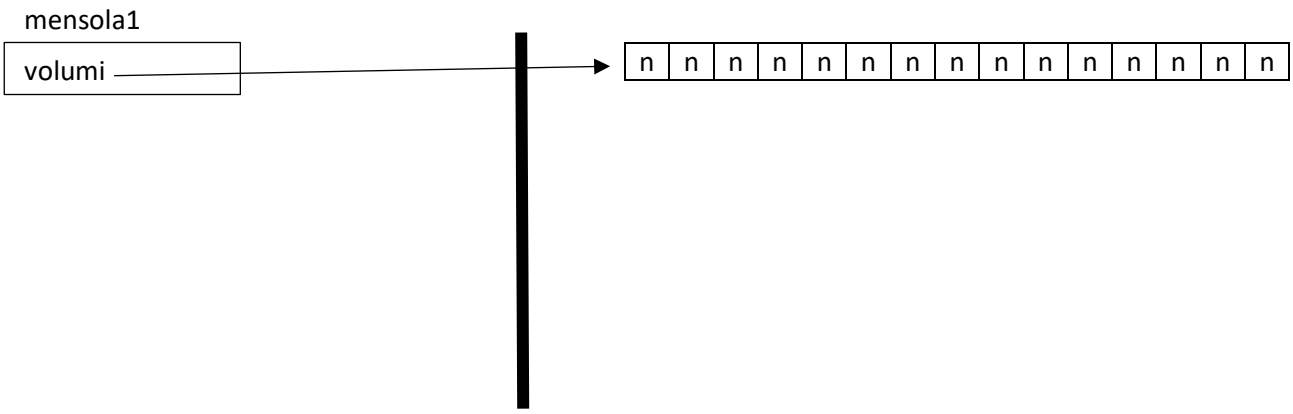



```
Libera (inserendo null) la posizione "posizione" e restituisce il numero della
posizione "liberata"
se la posizione indicata è<0 o >=  NUM_MAX_LIBRI (non valida) --> return -1
se la posizione è già vuota --> return -2
```

```
Restituisce il libro che si trova in una determinata posizione
se la posizione è vuota --> return null
se la posizione<0 o >= NUM_MAX_LIBRI (non valida) --> return null
```

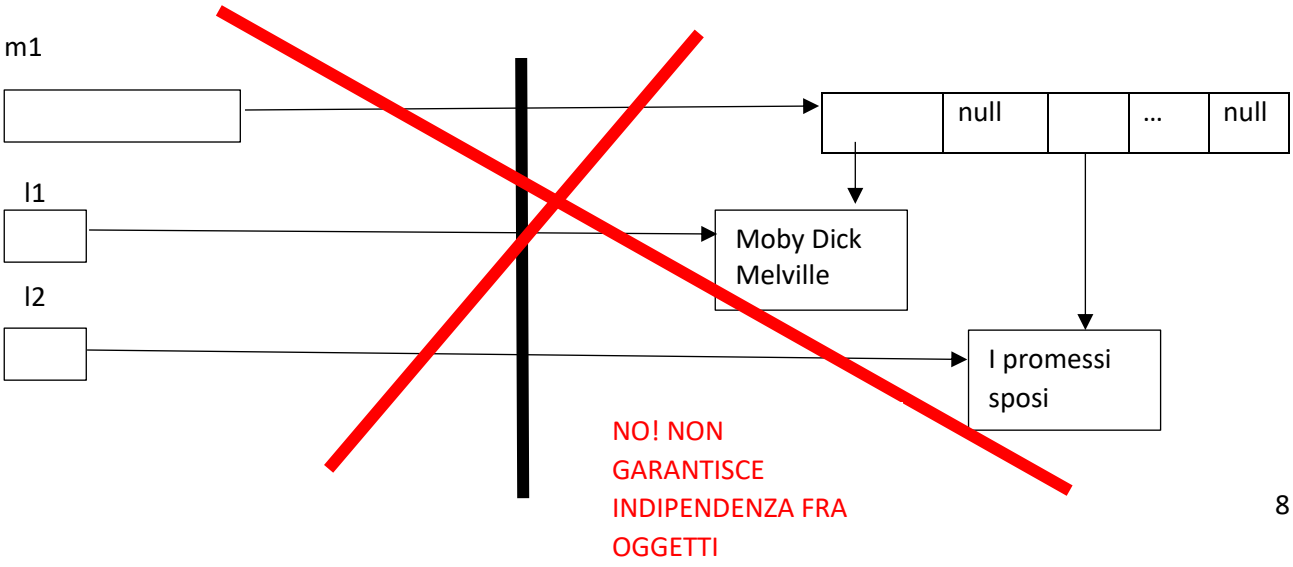
```
Aggiunge un libro alla mensola nella posizione indicata
se la posizione non esiste --> return -1
se la posizione è già occupata --> return -2
se il libro viene posizionato --> return posizione
```

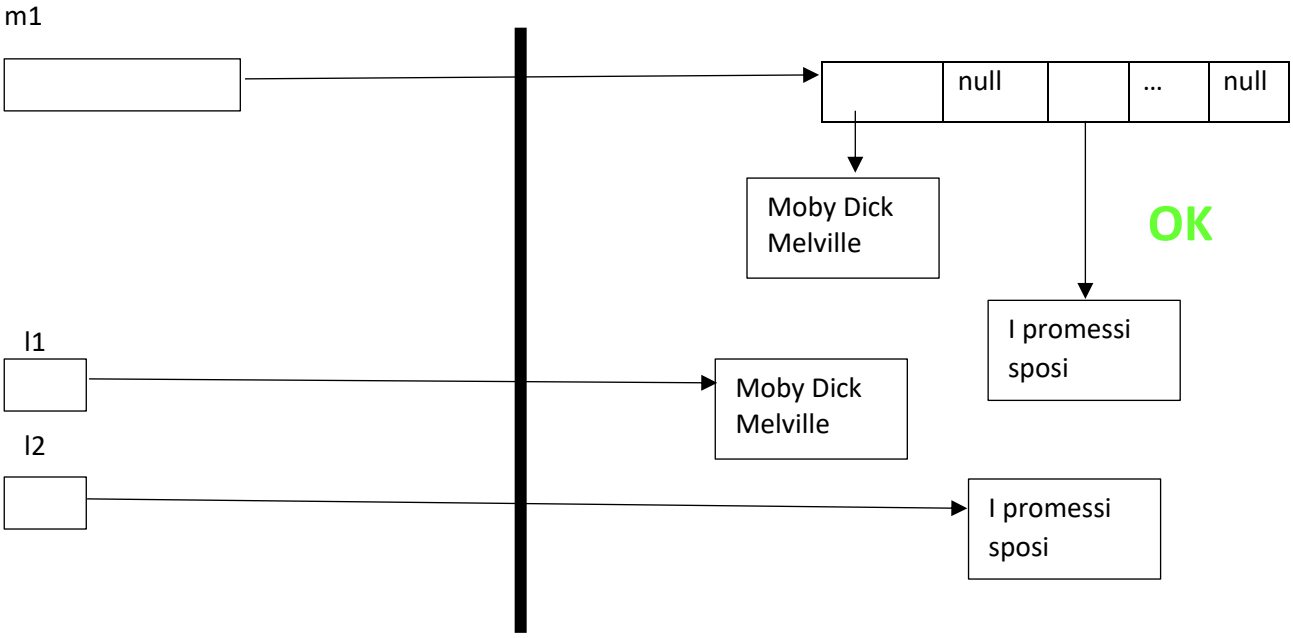
Si noti che quando si "costruisce" una mensola (quindi nel primo costruttore della classe Mensola) viene istanziata una array di 15 elementi "vuoti", ossia contenenti reference a null. Infatti la mensola inizialmente non contiene libri.




```
public Mensola()
{
   volumi=new Libro[NUM_MAX_VOLUMI];
}
```

Come già spiegato precedentemente, nella scrittura del metodo "setVolume (Libro volume, int posizione)" va posta attenzione al problema della dipendenza fra gli oggetti Libro, (con reference, ad esempio, "l1") istanziati nel main e passati come parametri a "setVolume", e l'oggetto mensola, anch'esso istanziato nel main, chiamato, ad esempio "m1". Ossia: una modifica al libro l1 dall' "esterno" della classe "mensola" provoca una modifica al libro dell'oggetto mensola. Può darsi che, in alcuni casi, questa situazione sia voluta dal progettista dei dati, ma generalmente non è così. Infatti Nella OOP è sempre opportuno mantenere la massima indipendenza fra le classi degli oggetti creati, ossia "la modifica di un oggetto mensola, o di un oggetto di cui essa è composta, deve avvenire attraverso metodi della classe mensola, non attraverso i metodi della classe "Libro"". Quindi per modificare un libro nella mensola, la procedura corretta è quella di invocare metodi su oggetti "che fanno parte della mensola. Non è corretto che una modifica su un oggetto (il libro "l1"), vada a modificare un altro oggetto (la mensola "m1"). Va mantenuta **l'indipendenza** fra le classi. Questa è una questione di sicurezza.





Per evitare questo problema è necessario che tutti i metodi della classe mensola che ricevono un parametro di tipo "Libro" (ad esempio "setVolume") o che restituiscono un oggetto di tipo "Libro" (ad esempio "getVolume"), agiscano su "delle copie" degli oggetti passati come parametri o restituiti. Il codice dei due metodi è dunque il seguente:

```
public int setVolume(Libro volume, int posizione)
{
   if (posizione<0 || posizione>=getNumMaxVolumi())
      return -1;      //Posizione non valida
   if (volumi[posizione]!=null)
      return -2;     //posizione già occupata
   else
   {
      volumi[posizione]=new Libro(volume);
      return posizione;
   }
}

public Libro getVolume(int posizione)
{
   if (posizione<0 || posizione>=getNumMaxVolumi())
      return null;      //Posizione non valida
   if (volumi[posizione]==null)
      return null;     //posizione vuota
    else
       return new Libro(volumi[posizione]);
}
```

Come regola generale, si può dire che ogni volta che due classi sono in associazione di composizione o aggregazione è più sicuro che i metodi della classe "tutto" agiscano su elementi "copia" della classe "parte".

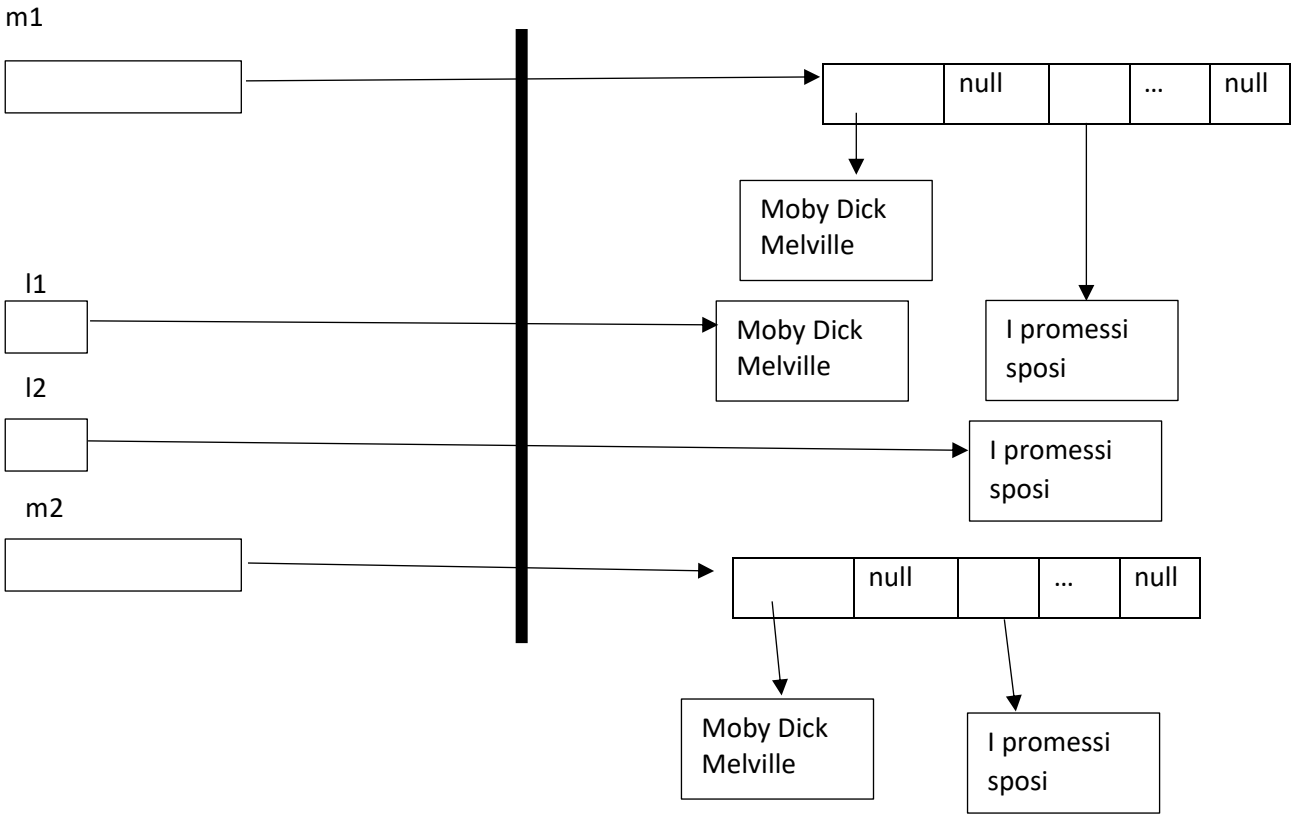
Infine, il metodo che consente di rimuovere il volume in una determinata posizione è il seguente:

```
public int rimuoviVolume(int posizione)
  {
    if (posizione<0 || posizione>=getNumMaxVolumi())
      return -1;      //Posizione non valida
    if (volumi[posizione]==null)
      return -2;     //posizione vuota
    else
    {
      volumi[posizione]=null;
      return posizione;
    }
  }
```

Per quanto riguarda il costruttore di copia della classe Mensola, grazie al fatto che il metodo "getVolume (int posizione)" restituisce **una copia** del libro in "posizione", la mensola ottenuta con il costruttore di copia sarà **indipendente** dalla mensola da cui si genera la copia. Il codice è il seguente:

```
public Mensola (Mensola m)
  {
    volumi=new Libro [getNumMaxVolumi()];
    for (int i=0;i<m.getNumMaxVolumi();i++)
    {
      if (m.getVolume(i)!=null)
      {
        volumi[i]=m.getVolume(i);    //restituisce UNA COPIA di ogni volume di m1
      }
    }
  }
```

Osservando ciò che accade nella memoria centrale risulta evidente l'indipendenza fra una mensola m1 ed una mensola m2 istanziata con il costruttore di copia applicata ad m1:



In m2 vi sono reference a COPIE dei libri di m1 quindi eventuali modifiche ai libri di m1 ora non influenzeranno più i libri di m2. In questo modo si è garantita l'indipendenza fra gli oggetti m1 ed m2.

Codice completo Classe Mensola

```
public class Mensola
{
  //atributi
  private Libro[] volumi;
  private static int NUM_MAX_VOLUMI=15;

  public Mensola()
  {
    volumi=new Libro[NUM_MAX_VOLUMI];
  }

  public Mensola (Mensola m)
  {
    volumi=new Libro[NUM_MAX_VOLUMI];
    for (int i=0;i<getNumMaxVolumi();i++)
    {
      volumi[i]=m.getVolume(i);
    }
  }
```

```
public static int getNumMaxVolumi()
{
   return NUM_MAX_VOLUMI;
}

public int getNumVolumi()
{
   int contatore=0;
   for (int i=0;i<getNumMaxVolumi();i++)
   {
      if (volumi[i]!=null)
         contatore++;
   }
   return contatore;
}


/*
Aggiunge un libro alla mensola nella posizione indicata
se la posizione non esiste --> return -1
se la posizione è già occupata --> return -2
se il libro viene posizionato --> return posizione
*/

public int setVolume(Libro volume, int posizione)
{
   if (posizione<0 || posizione>=getNumMaxVolumi())
      return -1;              //posizione non valida
   if (volumi[posizione]!=null)
      return -2;              //posizione occupata
   volumi[posizione]=new Libro(volume);
   return posizione;
}

/*
Restituisce il libro che si trova in una determinata posizione
se la posizione è vuota --> return null
se la posizione<0 o supera NUM_MAX_LIBRI (non valida) --> return null
*/
public Libro getVolume(int posizione)
{
   if (posizione<0 || posizione>=getNumMaxVolumi())
      return null;
   if (volumi[posizione]==null)
       return null;
   return new Libro(volumi[posizione]);
}
/*
```

```
    Libera (inserendo null) la posizione "posizione" e restituisce il numero della posizione
"liberata"
    se la posizione indicata è<0 o supera  NUM_MAX_LIBRI (non valida) --> return -1
    se la posizione è già vuota --> return -2
    */

    public int rimuoviVolume(int posizione)
    {
        if (posizione<0 || posizione>=getNumMaxVolumi())
            return -1;              //posizione non valida
        if (volumi[posizione]==null)
            return -2;              //posizione vuota
        volumi[posizione]=null;
        return posizione;
    }
}
```

Testiamo i vari metodi della classe mensola verificandone il corretto funzionamento. Per fare questo istanziamo nella main class due libri e una mensola e proviamo ad aggiungere e rimuovere i due libri in posizioni diverse. Verifichiamo anche l'indipendenza fra gli oggetti libro e l'oggetto mensola modificando uno dei due libri e constatando che la modifica non ha effetto sui libri della mensola. Si osservi che, grazie all'indipendenza degli oggetti, in una mensola si possono aggiungere e togliere diversi volumi dello stesso libro.

Per esercizio si aggiunga il metodo "presenzaTitolo" alla classe Mensola. Questo metodo riceve come parametro una stringa relativa ad un titolo e restituisce un valore booleano (true o false) che indica se un determinato titolo è presente o meno nella mensola.

Soluzione:

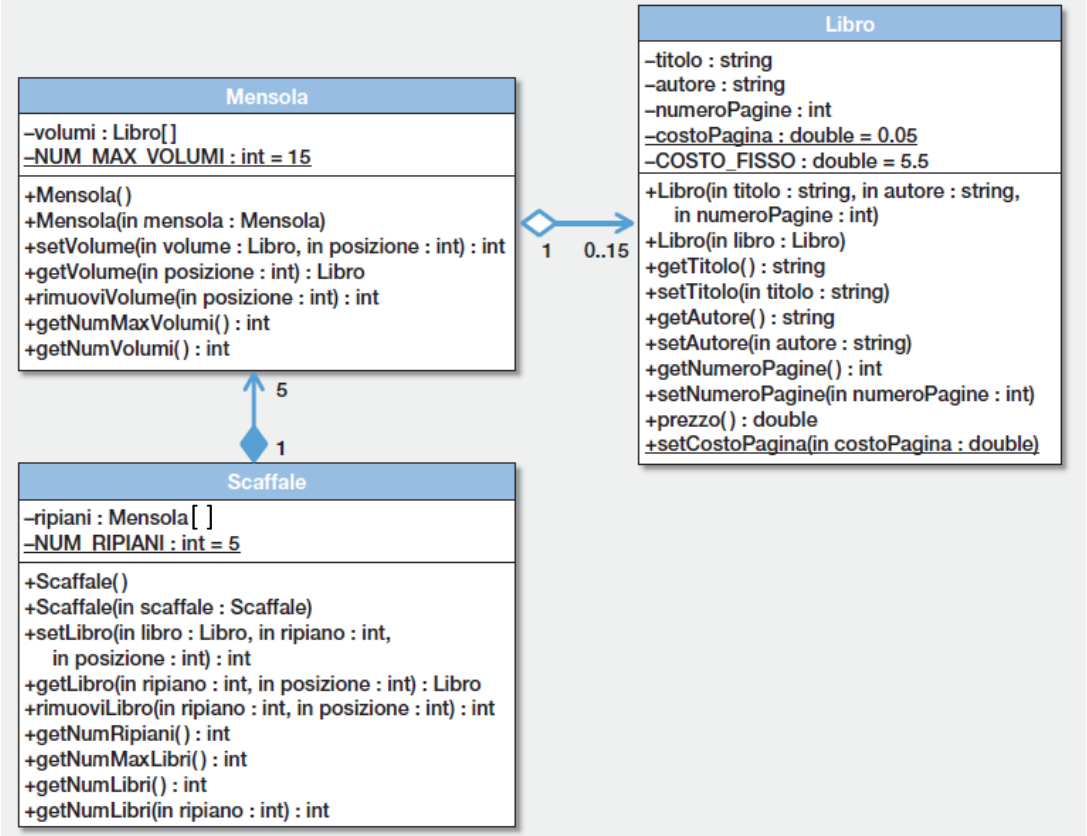
```
public boolean presenzaTitolo(String titolo)
    {
        for (int i=0;i<getNumMaxVolumi();i++)
        {
            if (volumi[i]!=null)                    //Attenzione!! serve questo!
            {
                if (volumi[i].getTitolo()==titolo)
                    return true;
            }
        }
        return false;
    }
```

Ora si modifichi il progetto Libreria aggiungendo una classe Scaffale come indicato nel seguente diagramma delle classi.

Link al video:

Video1: https://www.youtube.com/watch?v=DFJqTqNMznc&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=31

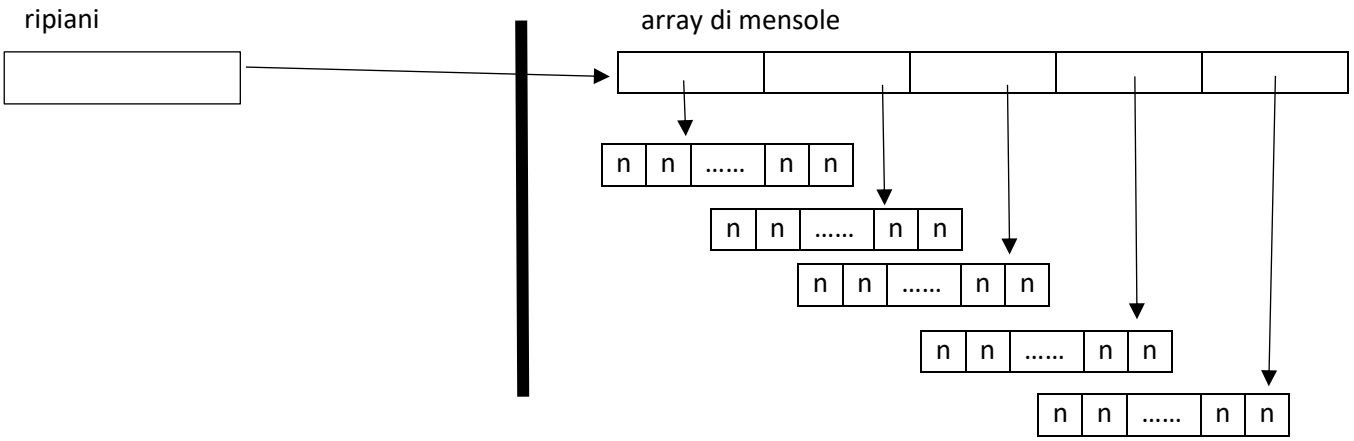
Video2: https://www.youtube.com/watch?v=FP06AjHUh1E&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=32



La classe scaffale è una composizione di 5 mensole (la classe “scaffale” è in associazione di composizione con la classe “mensola”).

Nel costruttore dello scaffale, non solo viene creato un array di oggetti “Mensola” (chiamata “ripiani[]”) con 5 elementi vuoti, ma poiché lo scaffale non può esistere senza mensole, vengono istanziati (ossia creati) anche i 5 oggetti mensola (vuoti) che costituiranno gli elementi dell’array ripiani[].

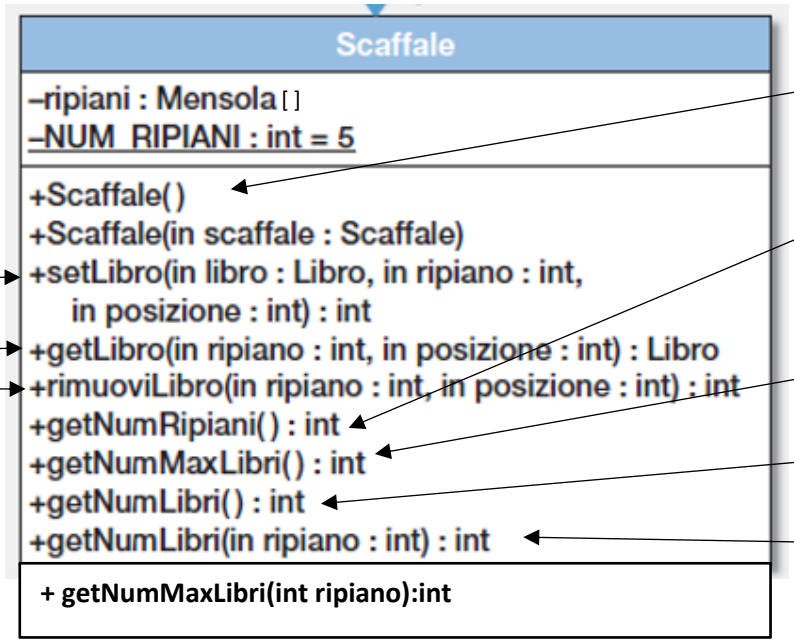
Il costruttore della mensola e quello dello scaffale sono quindi diversi. In quello della mensola si crea un array di 15 elementi **vuoti** (puntatori a null) destinati a “puntare” dei libri. Nel costruttore dello scaffale si crea un’array (ripiani) di 5 elementi di tipo mensola che **non puntano a null, anzi,** vengono creati (istanziati) **anche i 5 oggetti mensola (vuoti)** che compongono lo scaffale. Questo è perché, al momento della creazione, l’oggetto mensola non contiene nessun libro, mentre, al momento della creazione**, l’oggetto Scaffale contiene già esattamente 5 oggetti di tipo mensola**, che quindi vanno costruiti contemporaneamente alla “costruzione” dello scaffale.



oggetti mensola vuoti (con reference a null)

I metodi della classe scaffale per la gestione dei libri (setLibro, getLibro, rimuoviLibro) invocano i corrispondenti metodi della classe mensola.

Vediamo come vanno quindi implementati i metodi della classe Scaffale (codice disponibile in seguito)



Costruttore: Istanzia l'array ripiani costituito da 5 mensole vuote (anche esse devono essere istanziate).

Restituisce il numero di ripiani

Restituisce il numero massimo di volumi posizionabili nello scaffale

Restituisce il numero di volumi presenti

Restituisce il numero di volumi presenti nel ripiano. Se il ripiano non è valodo, return -1

```
Elimina il volume nella posizione "posizione" del ripiano "ripiano".
se il ripiano non è valido  --> return -3
se la posizione non è valida --> return -1
se già vuota --> return -2
se ok → return 0
```

```
Restituisce il volume nella posizione "posizione" del ripiano "ripiano".
se il ripiano non è valido, la posizione non è valida o vuota --> return null
se ok → ritorna l'oggetto libro
```

```
Inserisce il libro nella posizione "posizione" del ripiano "ripiano".
se il ripiano non è valido  --> return -3
se la posizione non è valida --> return -1
se la posizione non è vuota --> return -2
se ok → return 0
```

La classe “scaffale” non presenta la problematica della dipendenza poiché i ripiani di cui è composto sono nuovi oggetti istanziati con il costruttore Scafale e perché i metodi che agiscono sui libri invocano sempre i metodi della classe mensola (getVolume e setVolume) già adeguati rispetto al problema della dipendenza.

Codice completo classe Scaffale

```
public class Scaffale
{
  private Mensola[] ripiani;
  private static final int NUM_RIPIANI=5;

  public Scaffale()
  {
     ripiani=new Mensola[NUM_RIPIANI];

     for (int i=0;i<NUM_RIPIANI;i++)
     {
        ripiani[i]=new Mensola();
     }

  }

  public Scaffale(Scaffale s)
  {
      ripiani=new Mensola[NUM_RIPIANI];

      Libro l;
      for (int i=0;i<getNumRipiani();i++)
      {
         ripiani[i]=new Mensola();
         for(int j=0;j<ripiani[i].getNumMaxVolumi();j++)
         {
            if (s.getLibro(i, j)!=null)
            {
               l=s.getLibro(i, j);
               setLibro(l, i, j);
            }
         }
      }

  }
```

```
  /*
  Inserisce il libro nella "posizione" del “ripiano”.
se il ripiano non è valido  --> return -3
se la posizione non è valida --> return -1 se è occupata --> return -2
se ok  return 0


  */
  public int setLibro(Libro libro,int ripiano, int posizione)
  {
     int inserimentoOk;

     if (ripiano<0 || ripiano >=NUM_RIPIANI)
        return -3;

     inserimentoOk=ripiani[ripiano].setVolume(libro, posizione);
     if (inserimentoOk>=0)
        return 0;
     else
        return inserimentoOk;
  }

  /*
  Restituisce il libro nella "posizione" del “ripiano”.
se il ripiano non è valido, la posizione non è valida o vuota --> return null
se ok  ritorna l’oggetto libro

  */

  public Libro getLibro(int ripiano, int posizione)
  {
     if (ripiano<0 || ripiano>=NUM_RIPIANI)
        return null;
     return(ripiani[ripiano].getVolume(posizione));
  }

  /*
  Libera la "posizione" nel “ripiano”.
se il ripiano non è valido  --> return -3
se la posizione non è valida --> return -1 se già vuota --> return -2
se ok  return 0

  */

  public int rimuoviLibro(int ripiano, int posizione)
  {
     int rimozioneOk;
     if (ripiano<0 ||ripiano>=NUM_RIPIANI)
```

```
        return -3;   //ripiano non valido

    rimozioneOk=ripiani[ripiano].rimuoviVolume(posizione);
    if (rimozioneOk>=0)
        return 0;
    else
        return rimozioneOk;
}

public static int getNumRipiani()
{
    return NUM_RIPIANI;
}

public int getNumMaxLibri()
{
    int numMaxLibri=0;
    for(int i=0;i<getNumRipiani();i++)
    {
        numMaxLibri+=ripiani[i].getNumMaxVolumi();
    }
    return numMaxLibri;
}

public int getNumLibri()
{
    int numLibri=0;
    for(int i=0;i<getNumRipiani();i++)
    {
        numLibri+=ripiani[i].getNumVolumi();
    }
    return numLibri;
}

//se ripiano non valido --> return -1
public int getNumLibri(int ripiano)
{
    if (ripiano<0 || ripiano>=getNumRipiani())
        return -1;
    return ripiani[ripiano].getNumVolumi();

}

public int getNumMaxLibri(int ripiano)
{
    if (ripiano<0 || ripiano>=getNumRipiani())
        return -1;
    return ripiani[ripiano].getNumMaxVolumi();
```

```
    }

    public String toString()
    {
       String s="";

       if (getNumLibri()==0)
          s="Nessun libro presente";
       else
       {
        Libro libro;
        for(int i=0;i<getNumRipiani();i++)
        {
           s=s+"Ripiano "+i+":\n";
           for (int j=0;j<ripiani[i].getNumMaxVolumi();j++)
           {
              if (getLibro(i, j)!=null)
              {
                 libro=getLibro(i, j);
                 s=s+j+" --> "+libro.toString()+"\n";
              }
           }
        }
       }
      return s;
    }

   }
```

Test della classe “scaffale” nella MainClass():

1. Creare 3 libri l1 e l2, l3.
2. Creare uno scaffale s1.
3. Inserimento libri:
    a. Inserire il libro l1 nel ripiano 0 in posizione 10 (ok)
    b. Inserire il libro l2 nel ripiano 0 in posizione 0 m (ok)
    c. Creare un nuovo libro (l4) e inserirlo nel ripiano 1 posizione 2 (ok)
4. Test errori inserimento:
    a. Inserire l3 in ripiano 10 posizione 0 (“ripiano non valido”)
    b. Inserire l3 in ripiano 0 posizione 20 (“posizione non valida “)
    c. Inserire l3 in ripiano 0 posizione 10 (“posizione occupata”)
    d. Inserimento l3 in ripiano 1, posizione 0 (ok)
5. Test eliminazione
    a. Eliminazione del libro nel ripiano 2 posizione 9 (già vuota)
    b. Eliminazione del libro nel ripiano 9 posizione 9 (ripiano non valido)
    c. Eliminazione del libro nel ripiano 2 posizione 18 (posizione non valida)
    d. Eliminazione del libro nel ripiano 0 posizione 0 (ok)

6. Visualizzazione del contenuto di ogni ripiano della libreria con le seguenti informazioni:
    a. ripiano, posizione, titolo, autore, prezzo

7. Test costruttore di copia:
    a. creo un nuovo scaffale s2 con costruttore di copia, elimino da s1 il libro al ripiano 0 posizione 10, verifico che in s2 tale libro è ancora presente.

Array come parametri e valori di ritorno dei metodi di una classe

Come già visto in C gli array possono essere parametri passati ad un metodo. Si ricordi che in Java i parametri sono passati SEMPRE per VALORE e che gli array sono OGGETTI.

Diversamente da quanto abbiamo visto in C inoltre, gli array possono essere anche valori di ritorno dei metodi.

Vediamo due applicazioni di questi concetti.

- APPLICAZIONE 1: Aggiunta del metodo **ElencoTitoliAutore**: aggiungiamo alla classe **Scaffale** il metodo che consente di ottenere come **valore di ritorno** l'elenco dei TITOLI di un determinato autore presenti nello scaffale (attenzione: i TITOLI, non i libri, per questo motivo il valore di ritorno deve essere un array di String e non di oggetti Libro). (Per semplicità, non controlliamo l'eventuale presenza di titoli che si ripetono, se ci sono più volumi con lo stesso titolo lo scriviamo due volte).

  Codice:

```
public String[] elencoTitoliAutore(String autore)
{
   int numeroLibriAutore=0;
   Libro libro;

   //conto il numero di libri dell'autore presenti
   for (int i=0;i<getNumRipiani();i++)
   {
      for (int j=0;j<ripiani[i].getNumMaxVolumi();j++)
      {
         if (getLibro(i, j)!=null)
         {
            libro=getLibro(i, j);
            if (libro.getAutore().equalsIgnoreCase(autore))
               numeroLibriAutore++;
         }
      }
   }

   // se nn ci sono libri di quell'autore, return null
   if (numeroLibriAutore==0)
      return null;

   String[] elencoTitoliAutore=new String[numeroLibriAutore];

   int posizioneTitolo=0;

   //Assegno ad ogni elemento dell'array il titolo del libro
   for (int i=0;i<getNumRipiani();i++)
   {
      for (int j=0;j<ripiani[i].getNumMaxVolumi();j++)
```

```
        {
            if (getLibro(i, j)!=null)
            {
                libro=getLibro(i, j);
                if (libro.getAutore().equalsIgnoreCase(autore))
                {
                    elencoTitoliAutore[posizioneTitolo]=libro.getTitolo();
                    posizioneTitolo++;
                }
            }
        }
    }
    return elencoTitoliAutore;
}
```

Osservazioni importanti

1. L'array che viene restituito dal metodo elencoTitoliAutore viene **dimensionato dinamicamente**. Questo consente di non dover definire a priori la dimensione dell'array. La tecnica con cui progettare il metodo sarà quindi la seguente:

    a. Scansione di tutte le mensole e posizioni per "contare" quanti libri di quell'autore sono presenti, questo serve per istanziare poi l'array contenente i titoli con la dimensione corretta.
    b. Se sono presenti zero libri viene restituito null (si evita di istanziare l'array)
    c. Se sono presenti dei libri:
        i. si istanzia l'array che contiene i titoli di dimensione pari al numero di libri contati.
        ii. Scansione di tutte le mensole e posizioni per aggiungere all'array i titoli dei libri di quell'autore presenti.

2. Nel main, per testare la funzione, viene dichiarata una array "titoli[]" che però non viene istanziata, ma punterà all' array restituito dal metodo "elencoTitoliAutore".

3. Non è necessario che venga restituita UNA COPIA dei titoli perché l'array è di Stringhe, quindi oggetti immutabili (come tutti i tipi di dati nativi, proprio per questione di sicurezza). Nel caso si modificasse un titolo nell'array restituito, automaticamente ne verrebbe creata una copia così non si andrebbe a modificare il valore nell' array originale. Se invece il metodo avesse restituito una array di oggetti, sarebbe stato necessario che gli oggetti restituiti fossero delle copie (questione di sicurezza).

- APPLICAZIONE 2: Aggiunta, alla classe Mensola, di un costruttore Mensola (Libro[] elencoLibri) che accetta un array di libri come parametro e li posiziona, nella nuova mensola, nell'ordine in cui sono messi nell'elenco. Se il libri sono più di 15, quelli in eccesso vengono ignorati. Se l'array "elencoLibri" non contiene alcun libro viene istanziata una mensola vuota.

Codice:

```
public Mensola (Libro[] elencoLibri)
{
   volumi=new Libro[NUM_MAX_VOLUMI];

   int numeroLibri=0;
   //se elencoLibri non contiene libri
   if (elencoLibri.length==0)
      return;

   if (elencoLibri.length>getNumMaxVolumi())
      numeroLibri=getNumMaxVolumi();
   else
      numeroLibri=elencoLibri.length;

   for (int i=0;i<numeroLibri;i++)
   {
      if (elencoLibri[i]!=null)
      {
         volumi[i]=new Libro(elencoLibri[i]);
      }
   }

}
```

ESERCIZI

Realizzare in linguaggio java i software progettati in linguaggio UML (in TPS) relativi ai seguenti esercizi del libro di TPS. Nella realizzazione dei software si escluda la parte relativa al salvataggio su file dei dati, si realizzi, per ogni esercizio, oltre alle classi individuate, una classe "menu" che consenta all'utente di visualizzare gli oggetti creati e di eseguire le varie funzionalità fornite dal software.

Per ogni classe si aggiunga sempre un costruttore di copia.

Es 23 -- Catalogo computer (p.76 libro TPS)

ES 24 -- Container (p.76 libro TPS)

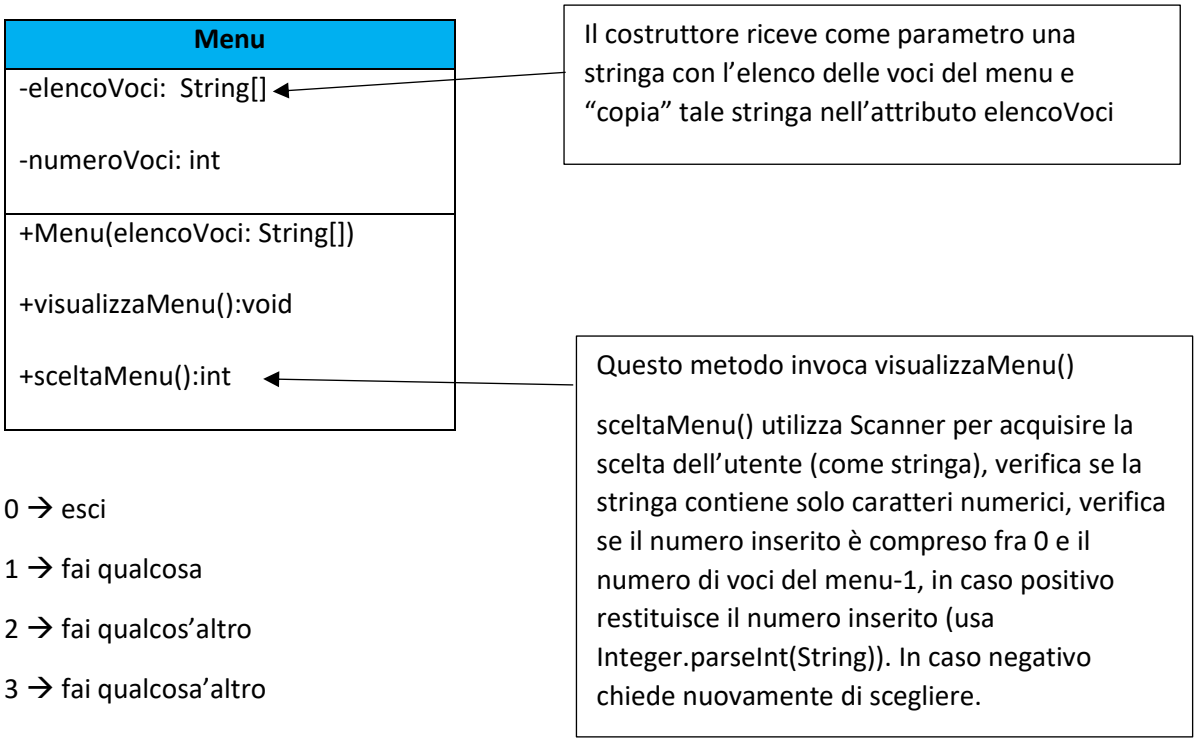
Es 28 -- Porto turistico (p.77 libro TPS)

In tutti gli esercizi sarebbe comodo utilizzare una classe Menu che visualizzi le voci di un menu numerico. La classe Menu deve esporre un metodo pubblico ("sceltaMenu") che consenta all'utente di scegliere una delle voci del menu e che svolga i controlli sulla validità della scelta effettuata da parte dell'utente.

Link ai video:

Video 1: https://www.youtube.com/watch?v=B3G4pMOAvvc&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=33

Video 2: https://www.youtube.com/watch?v=hhwVsQxJVo0&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=34



0 → esci

1 → fai qualcosa

2 → fai qualcos'altro

3 → fai qualcosa'altro

Codice della classe Menu:

```
public class Menu
{
   private String[] elencoVoci;
   private int numeroVoci;

   public Menu(String[] elencoVoci)
   {
      numeroVoci=elencoVoci.length;
      this.elencoVoci=new String[numeroVoci];
      for (int i=0;i<numeroVoci;i++)
         this.elencoVoci[i]=elencoVoci[i];
   }

   public void visualizzaMenu()
   {
      System.out.println("MENU:");
      for (int i=0;i<numeroVoci;i++)
         System.out.println(i+" --> "+this.elencoVoci[i]);
   }

   public int sceltaMenu()
   {
      String inputUtente;
      boolean inputUtenteOK=true;
      int sceltaUtente=0;



      do
      {
         visualizzaMenu();
         Scanner tastiera=new Scanner(System.in);
         System.out.println("Scelta -->");
         inputUtente=tastiera.nextLine();


         //verifico se l'inputUtente è un numero
         inputUtenteOK=true;
         for (int i=0;i<inputUtente.length();i++)
         {
            if (inputUtente.charAt(i)>='0' && inputUtente.charAt(i)<='9')
               i++;
            else
            {
               inputUtenteOK=false;
               break;
            }
```

```
        }

        //verifico se il numero inserito è compreso fra 0 e il numero di voci del menu -1
        if (inputUtenteOK)
        {
           sceltaUtente=Integer.parseInt(inputUtente);
           if (sceltaUtente<0 || sceltaUtente>numeroVoci-1)
              inputUtenteOK=false;
        }

        if (!inputUtenteOK)
        {
           System.out.println("Valore inserito non valido. Premi invio ed effettua nuovamente
la scelta");
           tastiera.nextLine();
        }

     }while(!inputUtenteOK);

     return sceltaUtente;
   }
}
```