Lezione 4

martedì 16 ottobre 2018 15:37

```
package it.unipi.abc;
import it.unipi.xyz.B;

public class A {
        :
        B b...
        :
}
```

A.java

```
package it.unipi.xyz;

public class B {
        :
}
```

B.java

```
it/
└→ unipi/
        ├→ abc/
        │      A.java
        │      A.class
        └→ xyz/
               B.java
               B.class
```

$ javac it/unipi/abc/A.java

classpath

Il classpath è un insieme di cartelle usate come punto di partenza nella ricerca delle classi (sia per JVM che per compilatore)

Negli esempi visti fino adesso il classpath è implicitamente la cartella corrente (.).

Per specificare il classpath usare opzione

-cp o -classpath

$ java -cp /usr/mario:/lib/myapp X

separatore
: unix
; win

la JVM cerca X.class in /usr/mario e se non lo trova in /lib/myapp

le cose funzionano nello stesso modo anche per javac

Se la classe appartiene a un package:

$ java -cp /usr/gino it.unipi.www.Y

la jvm cerca il file

/usr/gino/it/unipi/www/Y.class

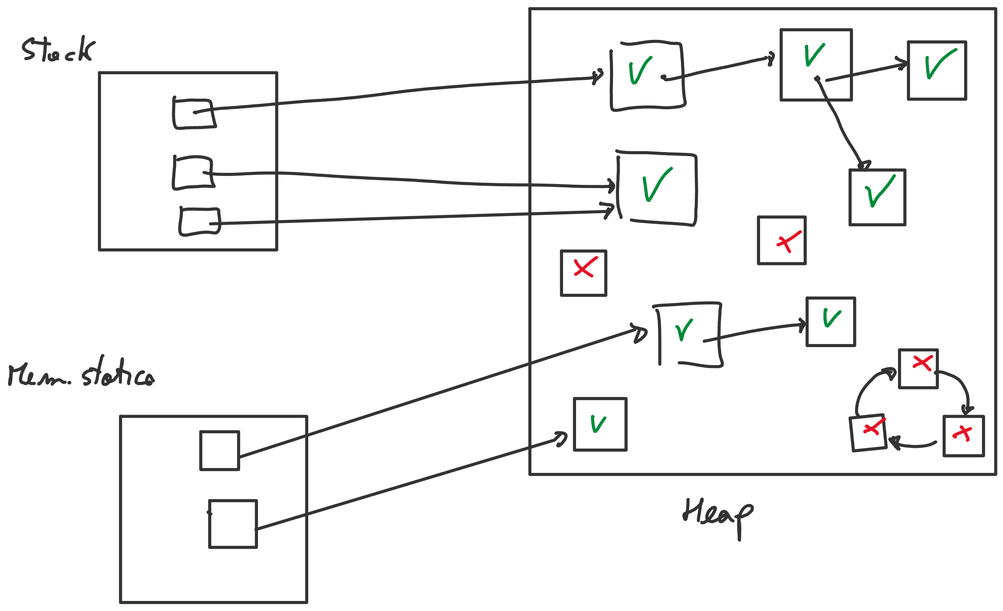
Per le classi delle librerie standard non è necessario specificare classpath (trovate automaticamente)

È possibile includere cartella corrente, per es:

$ java -cp .:/usr/gino ...

Gestione della memoria

Non è necessario deallocare la memoria relativa a oggetti non più utilizzati.
La deallocazione è automatica: un componente della JVM, il Garbage Collector, libera la memoria non più in uso.



Gli oggetti raggiungibili da stack e memoria statica vengono marcati ✓, quelli non marcati (x) vengono deallocati.

Il G.C. entra in funzione automaticamente

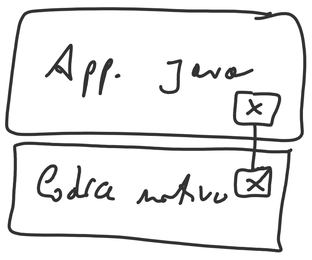
Se proprio vuole il programmatore può

```
System.gc()
```

Per ottenere notifiche quando un oggetto è in procinto di essere deallocato:

```
protected void finalize()
```

↑ Ridefinire questo metodo ereditato dalla classe Object



Java Native Interface

Array

Modello

La dimensione di un array è definita a tempo di esecuzione.
La dimensione di un array non può essere

cambiato.

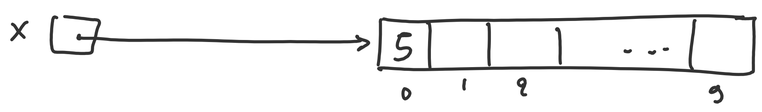
```
Tipo[]  nome;
```

```
int[]  x;
```

↑ crea un riferimento ad array di interi

```
x = new  int[10];
```

↑ crea un array di 10 int




```
x[0] = 5;
int z = x[3];
```

in generale

```
new Tipo[dimensione]
```

- Non esiste aritmetica dei puntatori
  `x++` NO!

- L'operatore di selezione con indice esegue a tempo di esecuzione un controllo di validità dell'indice

  Se sfora i limiti genera

  `ArrayIndexOutOfBoundsException`

- Ogni array ha un campo `length` che ne indica la dimensione

```
int[]  v = new int[100];
for (int i=0; i<v.length; i++)
    v[i] = i;
 :
```

- Il valore di default degli elementi è zero, false, carattere nullo.

Inizializzatori:

```
int[]  v1 = {10, 20, 30};
int[]  v2 = new int[] {40, 50};
```

Array di tipo riferimento:

```
public class Studente {
    String nome;
    double media;
     :
```

```
    void stampa() {
        System.out.println(nome + " " + media);
    }
}
```

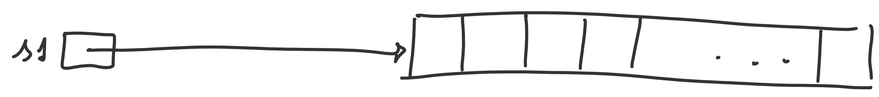
```
Studente[] s1;
```

↖ riferimento ad array di Studente

```
s1 = new Studente[10];
```

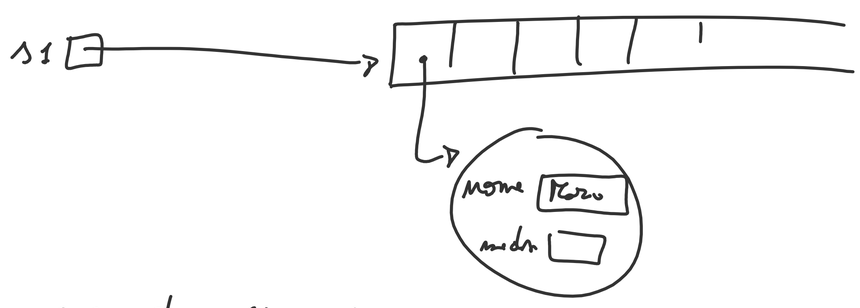
↖ array di riferimenti a Studente

Non ci sono oggetti di tipo Studente




```
s1[0].stampa(); // errore: NullPointerException
```

```
s1[0] = new Studente("Mario");
```




```
s1[0].stampa(); // ok
```

Inizializzazione:

```
Studente[] s2 = { new Studente("Mario"),
                  new Studente("Gino")};

Studente[] s3 = new Studente[]{
                  new Studente("Pino"),
                  new Studente("Furio")};
```

Stringhe

- Non sono array di caratteri terminati dal carattere null
- istanze della classe String

```
String a1 = "abc";
String a2 = new String("def");
char[] v = {'x','y','z'};
String a3 = new String(v);
```

- le stringhe sono immutabili
- Se necessario fare (numerose) operazioni di modifica usare la classe StringBuffer

```
String s = "abc";
s = s + "def";
```

↖ crea una nuova stringa "abcdef"

- Concatenazione con +

```
String r = "Rambo" + 1 + 2; // Rambo12
String z = 1 + 2 + "Rambo"; // 3 Rambo
```

- Alcuni metodi:

```
public String substring(int begin, int end)
String w = "Smiles";
String y = w.substring(1,5); // mile
```

- metodo length()

```
String z = "abc";
int l = z.length(); // l vale 3
```

- Per conoscere carattere i-esimo:

```
char charAt(int i)

char c = "Pippo".charAt(2); // c vale 'p'
```

- Confronto

usare il metodo equals

Non usare ==

== confronta i riferimenti, restituisce
true se puntano allo stesso oggetto
false altrimenti

```
public boolean equals(Object o)
```

restituisce true solo se
- è una stringa
- non vale null
- contiene gli stessi caratteri dell'oggetto implicito

```
String s1 = "pippo";
String s2 = "pluto";
boolean b1 = s1.equals(s2); // false
boolean b2 = "pluto".equals(s2); // true

String s3 = "pippo";
boolean b3 = s1.equals(s3); // true
boolean b4 = s1 == s3; ?
```

Esercizio

Biblioteca
Una biblioteca gestisce un insieme di libri. Ogni libro è caratterizzato da un titolo, un autore e una data di pubblicazione. Un autore è caratterizzato da un nome e un cognome. Realizzare le classi Data, Libro, Autore e Biblioteca in modo che sia possibile compiere le azioni elencate di seguito.

Classe Data:
• Data(int giorno, int mese, int anno): crea un oggetto Data dalle caratteristiche

• Data(int giorno, int mese, int anno): crea un oggetto Data dalle caratteristiche specificate.
• int confronta(Data d): confronta l'oggetto Data a cui viene applicato con la data d, e restituisce un numero negativo, zero, o un numero positivo rispettivamente quando l'oggetto precede, è uguale a, o segue la data d.

Classe Autore:
• Autore(String n, String c): crea un oggetto Autore con nome n e cognome c.
• boolean uguale(Autore a): restituisce true se il nome e il cognome dell'oggetto implicito sono uguali a quelli dell'oggetto a; il valore null per il nome e/o il cognome dell'autore a può essere usato come valore "jolly"; in tutti gli altri casi restituisce false.

Classe Libro:
• Libro(String t, Autore a, Data d): crea un oggetto Libro dalle caratteristiche specificate.
• String getTitolo(): restituisce il titolo del libro.
• Autore getAutore(): restituisce l'autore del libro.
• Data getDataPub(): restituisce la data di pubblicazione del libro.

Classe Biblioteca:
• Biblioteca(int n): costruttore che crea un oggetto Biblioteca in grado di gestire al più n libri.
• boolean aggiungiLibro(Libro l): aggiunge il libro l a quelli gestiti dalla biblioteca; restituisce true se è possibile aggiungere il libro, false altrimenti (la biblioteca contiene già il numero massimo di libri).
• Libro[] cercaPerAutore(Autore a): restituisce i libri che hanno l'autore specificato.
• Libro[] cercaPerCognome(String c): restituisce i libri il cui autore ha il cognome specificato.
• Libro[] elenco(): restituisce l'elenco di tutti i libri.
• Libro[] cercaRecenti(Data d): restituisce l'elenco dei libri la cui data di pubblicazione è più recente di d.
• boolean elimina(String t): se un libro con titolo t è presente nella biblioteca lo elimina e restituisce il valore true, false altrimenti.

```
package cap6.biblio;

public class Data {

  private int giorno;
  private int mese;
  private int anno;

  public Data(int g, int m, int a) {
    giorno = g;
    mese = m;
    anno = a;
  }

  public int confronta(Data d) {
    if (anno != d.anno)
      return anno - d.anno;
    if (mese != d.mese)
      return mese - d.mese;
    return giorno - d.giorno;
  }

  public void stampa() {
    System.out.println(giorno + "/" + mese + "/" + anno);
  }
}
```

```
package cap6.biblio;

public class Libro {

  private String titolo;
  private Autore autore;
  private Data dataPub;

  public Libro(String t, Autore a, Data d)
  {
    titolo = t;
    autore = a;
    dataPub = d;
  }

  public Autore getAutore() {
    return autore;
  }

  public String getTitolo() {
    return titolo;
  }

  public Data getDataPub() {
    return dataPub;
  }
}
```

```
package cap6.biblio;

public class Autore {

  private String nome;
  private String cognome;

  public Autore(String n, String c) {
    nome = n;
    cognome = c;
  }

  public boolean uguale(Autore a) {
    if ((a.nome == null) && (a.cognome == null))
      return true;
    if (a.nome == null)
      return cognome.equals(a.cognome);
    if (a.cognome == null)
      return nome.equals(a.nome);
    return nome.equals(a.nome) &&
          cognome.equals(a.cognome);
  }

  public void stampa() {
    System.out.println(nome + " " + cognome);
  }
}
```

```
package cap6.biblio;

public class Biblioteca {

  private Libro[] lib;
  private int numLibri;

  public Biblioteca(int n) {
    lib = new Libro[n];
  }

  public boolean aggiungiLibro(Libro l) {
    if (numLibri < lib.length) {
      lib[numLibri++] = l;
      return true;
    }
    else
      return false;
  }

  public Libro[] cercaPerAutore(Autore a) {
    int n = 0;
    for (int i = 0; i < numLibri; i++)
      if (lib[i].getAutore().uguale(a))
        n++;
    if (n == 0)
      return null;
    Libro[] risul = new Libro[n];
    n = 0;
    for (int i = 0; i < numLibri; i++)
      if (lib[i].getAutore().uguale(a))
        risul[n++] = lib[i];
    return risul;
  }

  public Libro[] cercaPerCognome(String c) {
    return cercaPerAutore(new Autore(null, c));
  }

  public Libro[] elenco() {
    return cercaPerAutore(new Autore(null, null));
  }

  public Libro[] cercaRecenti(Data d) {
    int n = 0;
    for (int i = 0; i < numLibri; i++)
      if (lib[i].getDataPub().confronta(d) >= 0)
        n++;
```

```
    if (n == 0)
      return null;
    Libro[] risul = new Libro[n];
    n = 0;
    for (int i = 0; i < numLibri; i++)
      if (lib[i].getDataPub().confronta(d) >= 0)
        risul[n++] = lib[i];
    return risul;
  }

  public boolean elimina(String t) {
    for (int i = 0; i < numLibri; i++)
      if (lib[i].getTitolo().equals(t)) {
        lib[i] = lib[--numLibri];
        lib[numLibri] = null;
        return true;
      }
    return false;
  }

}
```