ArrayList

Dal materiale di Stefano Ferretti

s.ferretti@unibo.it

Vector e ArrayList

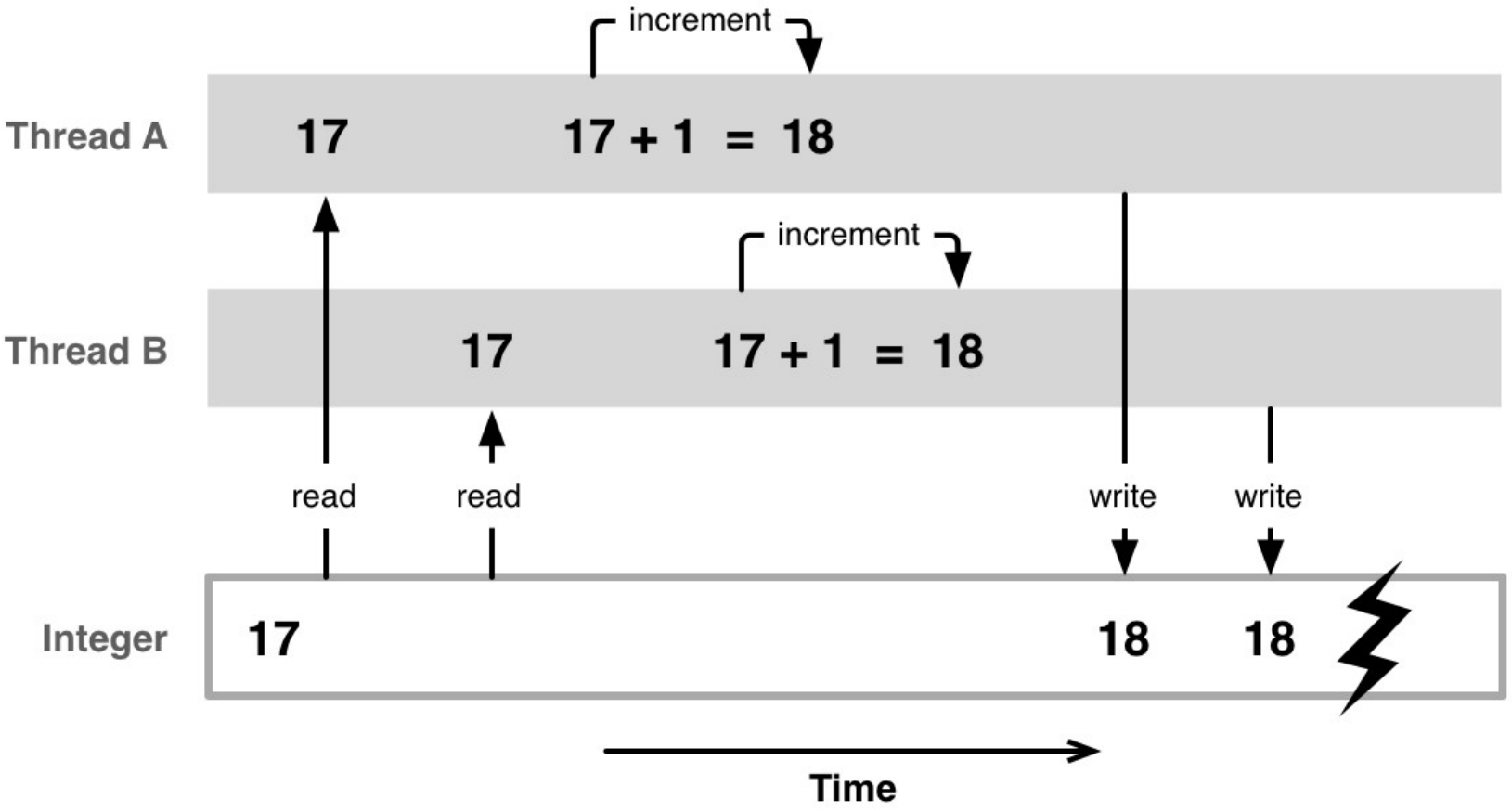
- Abbiamo visto le principali caratteristiche e operazioni su array e liste concatenate
- La libreria **standard** di Java fornisce classi che definiscono "*array evoluti*", ad es.
  - `java.util.Vector`
  - `java.util.ArrayList`
- Queste classi vanno importate esplicitamente
  - Es., `import java.util.ArrayList`

Vector e ArrayList

- **`Vector`** e **`ArrayList`** sono molto simili, differiscono nella gestione della **concorrenza**
  - Cioè, accessi *simultanei* a **memoria condivisa** da parte di diversi **thread** (sottoprocessi della JVM)
  - **Multithreading** potenzialmente problematico con scritture concorrenti (*race conditions*)

- **`Vector`** è protetto da accessi simultanei (**sincronizzato**) mentre **`ArrayList`** **no**

Race Conditions



* From **http://opensourceforgeeks.blogspot.com/2014/01/race-condition-synchronization-atomic.html**

Vector e ArrayList

- Se non si gestiscono thread diversi possiamo tranquillamente usare **`ArrayList`**
- Altrimenti è meglio usare **`Vector`**
  - Ad es. le applicazioni con interfaccia grafica (**GUI**) sono multithreaded
- Per ora ci concentriamo su **`ArrayList`**, ma quello che vedremo vale anche per **`Vector`**

ArrayList

- Un oggetto di **`ArrayList`** rappresenta una **lista sequenziale** di **oggetti**
- Gli elementi della sequenza sono **indicizzati** a partire dal primo
- Come per gli array gli indici partono da **0** e si può **accedere direttamente** ad un elemento tramite il suo indice nell'ArrayList

ArrayList

- Si può sempre inserire un nuovo elemento **in coda** a un **`ArrayList`** col metodo **`add`**
- **Non** c'è limite (a parte la memoria fisica della macchina) alla **dimensione** di un'**`ArrayList`**
- La classe alloca **automaticamente** nuovo spazio quando ne ha bisogno
  - In modo **trasparente** all'utente

Metodi di ArrayList

`public ArrayList<`*tipo_base*`>(int capacitaIniziale)`
Crea una lista vuota di elementi di tipo *tipo_base* e con una determinata `capacitaIniziale`. *tipo_base* deve essere una classe; non può essere un tipo primitivo come `int` o `double`. Quando la lista ha bisogno di incrementare la propria capacità, la capacità raddoppia.

`public ArrayList<`*tipo_base*`>()`
Si comporta come il costruttore precedente, ma la capacità iniziale è pari a 10.

`public boolean add(`*tipo_base* `nuovoElemento)`
Inserisce `nuovoElemento` alla fine di questa lista e incrementa la dimensione della lista di 1 unità. Se necessario incrementa la capacità della lista. Restituisce vero se l'inserimento avviene con successo.

`public void add(int indice,` *tipo_base* `nuovoElemento)`
Inserisce `nuovoElemento` nella posizione `indice` di questa lista. Per fare spazio al nuovo elemento, sposta gli elementi successivi incrementando il loro indice di 1 unità. La dimensione della lista viene incrementata di 1. Se necessario incrementa la capacità della lista. Se `indice < 0` o se `indice ≥ size()` scatena un errore di indice fuori dai limiti (*index out of bounds*).

`public` *tipo_base* `get(int indice)`
Restituisce l'elemento alla posizione `indice` di questa lista. Se `indice < 0` o se `indice ≥ size()` scatena un errore di indice fuori dai limiti (*index out of bounds*).

`public` *tipo_base* `set(int indice,` *tipo_base* `elemento)`
Sostituisce l'elemento alla posizione `indice` di questa lista con `elemento`. Restituisce l'elemento sostituito. Se `indice < 0` o se `indice ≥ size()` scatena un errore di indice fuori dai limiti (*index out of bounds*).

Metodi di ArrayList

`public` *tipo_base* `remove(int indice)`
Rimuove e restituisce l'elemento alla posizione `indice` di questa lista. Sposta gli elementi che sono nelle posizioni successive decrementando il loro indice di 1 unità. Decrementa la dimensione della lista di 1 unità. Se `indice < 0` o se `indice ≥ size()` scatena un errore di indice fuori dai limiti (*index out of bounds*).

`public boolean remove(Object elemento)`
Rimuove la prima occorrenza di `elemento` in questa lista e sposta gli elementi successivi decrementando il loro indice di 1 unità. Decrementa la dimensione della lista di 1 unità. Restituisce vero se `elemento` è stato rimosso; altrimenti restituisce falso e non altera la lista.

`public void clear()`
Rimuove tutti gli elementi da questa lista.

`public int size()`
Restituisce il numero di elementi di questa lista.

`public boolean contains(Object elemento)`
Restituisce vero se `elemento` è in questa lista; altrimenti restituisce falso.

`public int indexOf(Object elemento)`
Restituisce l'indice della prima occorrenza di `elemento` in questa lista. Restituisce -1 se l'elemento non è nella lista.

`public boolean isEmpty()`
Restituisce vero se questa lista è vuota; altrimenti restituisce falso.

Generics

- In Java è possibile istanziare una collezione **generica** in base al **tipo** dei suoi elementi
- Il **tipo base** è sempre un (riferimento a un) **oggetto** e va inserito tra parentesi angolari subito dopo **`ArrayList`**
  - **No** tipi primitivi, es. **`ArrayList<int> A; //Errore!`**
- **`ArrayList<T>`** contiene solo oggetti di tipo **`T`**

```
ArrayList<BankAccount> accounts = new ArrayList<BankAccount>();
accounts.add(new BankAccount(1001));
accounts.add(new BankAccount(1015));
accounts.add(new BankAccount(1022));
```

Generics

- I tipi generici rendono una classe **parametrica** in base a un certo tipo **T**
  - I metodi e costruttori sono definiti in termini di **T**
- In questo modo, la classe è definita **una** sola volta ma può essere **instanziata** con diversi tipi base
  - ArrayList<Cane>, ArrayList<Persona>, ArrayList<Auto>, …
  - Non c'è necessità di effettuare **cast**
- Il **compilatore** verifica *staticamente* la compatibilità del tipo base

Size and capacity

- Di default, la **capacità** iniziale di un ArrayList è **10**
  - **Capacity** = dimensione array "*sottostante*" usato da ArrayList, è sempre **≥** della **size** = num. *effettivo* di elementi della lista
  - Si può settare una capacità diversa nel costruttore

- **NOTA**: Il **costruttore** di ArrayList crea sempre una sequenza **vuota** (`size == 0`) qualunque sia la capacità (a meno che sia creato da un'altra collezione di elementi)
  - La capacità può essere **modificata** dopo la creazione
  - **Non** esistono metodi che **ritornano** la capacità
  - https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/ArrayList.html

Lettura di elementi

- Il metodo **`get`** ritorna un elemento avente il tipo base specificato, ad es.

```
BankAccount anAccount = accounts.get(2);
   // fornisce il terzo elemento del vettore
```

  - Analogo a **A[i]** per un array A

- Come per gli array, **attenzione** agli indici!

```
int n = accounts.size();
anAccount = accounts.get(n); // Errore!!!
/* Gli indici validi vanno da 0 a n-1 */
```

Modifica e inserzione

- Con **`set`** si **sovrascrive** l'elemento in una certa posizione **`i`** con un altro elemento

```
BankAccount anAccount = new BankAccount(1729);
accounts.set(2, anAccount);
```

- Con **`add`** è *anche* possibile **inserire** un elemento in una posizione **arbitraria `i`**
  - Gli elementi dalla posizione **`i`** in poi "*scalano*" a destra di una posizione (*shifting*)

```
accounts.add(i, x)
```

Cancellazione

- **`clear`** rimuove **tutti** gli elementi dalla lista
- **`remove`** ha 2 versioni **overloaded**:
  - **`remove(int i)`** Rimuove l'elemento in posizione **`i`**
  - **`remove(Object o)`** Rimuove la **prima** occorrenza dell'oggetto **`o`** dalla lista
    - Se viene cancellato un elemento, tutti i successivi "*scalano*" di una posizione
- **`removeAll`**, **`removeIf`**, **`removeRange`** …
  - **https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/ArrayList.html**

Ricerca

- **`contains(Object o)`** ritorna **true** *sse* **`o`** è presente nella lista

- **`indexOf(Object o)`** ritorna l'**indice** della **prima occorrenza** dell'oggetto **`o`** nella lista
- **`lastIndexOf(Object o)`** ritorna l'**indice** dell'**ultima occorrenza** dell'oggetto **`o`** nella lista
  - Se **`o`** non è presente, ritornano **-1**

- Es. se X è un ArrayList avente elementi **[1, 2, 2, 0, 1]** allora:
  - X.contains(**0**) = **?**, X.indexOf(**0**) **=** **?**, X.lastIndexOf(**0**) = **?**
  - X.contains(**1**) = **?**, X.indexOf(**1**) = **?**, X.lastIndexOf(**1**) = **?**
  - X.contains(**2**) = **?**, X.indexOf(**2**) = **?**, X.lastIndexOf(**2**) = **?**
  - X.contains(**3**) = **?**, X.indexOf(**3**) **=** **?**, X.lastIndexOf(**3**) = **?**

Ricerca

- **`contains(Object o)`** ritorna **true** *sse* **`o`** è presente nella lista

- **`indexOf(Object o)`** ritorna l'**indice** della **prima occorrenza** dell'oggetto **`o`** nella lista
- **`lastIndexOf(Object o)`** ritorna l'**indice** dell'**ultima occorrenza** dell'oggetto **`o`** nella lista
  - Se **`o`** non è presente, ritornano **-1**

- Es. se X è un ArrayList avente elementi **[1, 2, 2, 0, 1]** allora:
  - X.contains(**0**) = **true**, X.indexOf(**0**) **=** X.lastIndexOf(**0**) = **3**
  - X.contains(**1**) = **true**, X.indexOf(**1**) = **0**, X.lastIndexOf(**1**) = **4**
  - X.contains(**2**) = **true**, X.indexOf(**2**) = **1**, X.lastIndexOf(**2**) = **2**
  - X.contains(**3**) = **false**, X.indexOf(**3**) **=** X.lastIndexOf(**3**) = **-1**

Iterare un ArrayList

Accesso sequenzale

- L'iterazione su ArrayList è analoga a quanto visto per gli array

- Es. stampa tutti gli elementi di un ArrayList di stringhe uno sotto l'altro:

```
ArrayList<String> listaStringhe = …
for (int i = 0; i<listaStringhe.size(); i++)
  System.out.println(listaStringhe.get(i));
```

```
import java.util.ArrayList;
import java.util.Scanner;

public class ArrayListDemo {

    public static void main(String[] args) {
        ArrayList<String> listaDelleCoseDaFare = new ArrayList<String>();

        System.out.println("Inserisci gli elementi per" +
                           " la lista quando richiesto.");
        boolean fatto = false;
        Scanner tastiera = new Scanner(System.in);

        while (!fatto) {
            System.out.println("Inserisci un elemento:");
            String elemento = tastiera.nextLine();
            listaDelleCoseDaFare.add(elemento);
            System.out.print("Altri elementi per la lista? ");
            String risposta = tastiera.nextLine();

            if (!risposta.equalsIgnoreCase("si"))
                fatto = true;
        }

        System.out.println("La lista contiene:");
        int dimensioneLista = listaDelleCoseDaFare.size();

        for (int posizione = 0; posizione < dimensioneLista; posizione++)
            System.out.println(listaDelleCoseDaFare.get(posizione));
    }
}
```

Es. carico e stampo una lista di stringhe

Esempio di output

```
Inserisci gli elementi per la lista quando richiesto.
Inserisci un elemento:
Comprare il latte
Altri elementi per la lista? si
Inserisci un elemento:
Lavare l'auto
Altri elementi per la lista? si
Inserisci un elemento:
Fare il compito
Altri elementi per la lista? no
La lista contiene:
Comprare il latte
Lavare l'auto
Fare il compito
```

Esempio: BankAccount.java

```
01: /**
02:    Un conto corrent ha un certo saldo che può essere modificato
03:    da depositi e prelievi successivi
04: */
05: public class BankAccount
06: {
07:    /**
08:       Costruisce un conto bancario con saldo uguale a zero.
09:       @param anAccountNumber il numero di questo conto bancario
10:    */
11:    public BankAccount(int anAccountNumber)
12:    {
13:       accountNumber = anAccountNumber;
14:       balance = 0;
15:    }
16:
```

Esempio: BankAccount.java

```
17:    /**
18:       Costruisce un conto bancario con un saldo iniziale.
19:       @param anAccountNumber il numero di questo conto bancario
20:       @param initialBalance il saldo iniziale
21:    */
22:    public BankAccount(int anAccountNumber,
                          double initialBalance)
23:    {
24:       accountNumber = anAccountNumber;
25:       balance = initialBalance;
26:    }
27:
28:    /**
29:       Restituisce il numero di conto del conto bancario.
30:       @return il numero di conto
31:    */
32:    public int getAccountNumber()
33:    {
34:       return accountNumber;
35:    }
```

Segue

Esempio: BankAccount.java

```
36:
37:    /**
38:       Versa denaro nel conto bancario.
39:       @param amount l'importo da versare
40:    */
41:    public void deposit(double amount)
42:    {
43:       double newBalance = balance + amount;
44:       balance = newBalance;
45:    }
46:
47:    /**
48:       Preleva denaro dal conto bancario.
49:       @param amount l'importo da prelevare
50:    */
51:    public void withdraw(double amount)
52:    {
53:       double newBalance = balance - amount;
54:       balance = newBalance;
```

Esempio: BankAccount.java

```
55:    }
56:
57:    /**
58:       Ispeziona il valore del saldo attuale del conto.
59:       @return il saldo attuale
60:    */
61:    public double getBalance()
62:    {
63:       return balance;
64:    }
65:
66:    private int accountNumber;
67:    private double balance;
68: }
```

ArrayListTester.java

```
01: import java.util.ArrayList;
02:
03: /**
04:    Questo programma collauda la classe ArrayList.
05: */
06: public class ArrayListTester
07: {
08:    public static void main(String[] args)
09:    {
10:       ArrayList<BankAccount> accounts
11:             = new ArrayList<BankAccount>();
12:       accounts.add(new BankAccount(1001));
13:       accounts.add(new BankAccount(1015));
14:       accounts.add(new BankAccount(1729));
15:       accounts.add(1, new BankAccount(1008));
16:       accounts.remove(0);
```

ArrayListTester.java

```
17:
18:       System.out.println("size=" + accounts.size());
19:       BankAccount first = accounts.get(0);
20:       System.out.println("first account number="
21:             + first.getAccountNumber());
22:       BankAccount last = accounts.get(accounts.size()-1);
23:       System.out.println("last account number="
24:              + last.getAccountNumber());
25:    }
26: }
```

BankAccount.java

Output:

```
Size: 3
First account number: 1008
Last account number: 1729
```

Popolare un ArrayList

- Inserire uno ad uno tutti gli elementi di un ArrayList è scomodo, soprattutto quando sono noti *a priori*

- E' desiderabile utilizzare qualcosa simile all'inizializzazione degli array, es.

```
String[] data = {“uno”, “due”, “tre”};
int[] numeri = {-1, 6, 4, -3, 8};
```

Popolare un ArrayList

- Il metodo **`asList`** di **`java.util.Arrays`** ritorna la lista corrispondente alla sequenza specificata

```
ArrayList<String> arrayList = new ArrayList<String>(
  Arrays.asList("uno", "due", "tre")
);
System.out.println(arrayList);
// Output: [uno, due, tre]
```

Stampare un ArrayList

- La stampa di un **ArrayList<T>** funziona come ci si aspetta se la stampa è "*ben-definita*" per il tipo base **T**
  - Per gli array non è così
- Se X è un ArrayList avente di **n** elementi $\mathbf{X_1, \ldots, X_n}$ allora **System.out.println(X)** stampa su std output la stringa:

  $$\mathbf{[S_1, \ldots, S_n]}$$

  dove $\mathbf{S_i = X_i.toString()}$ per i = 1, …, n

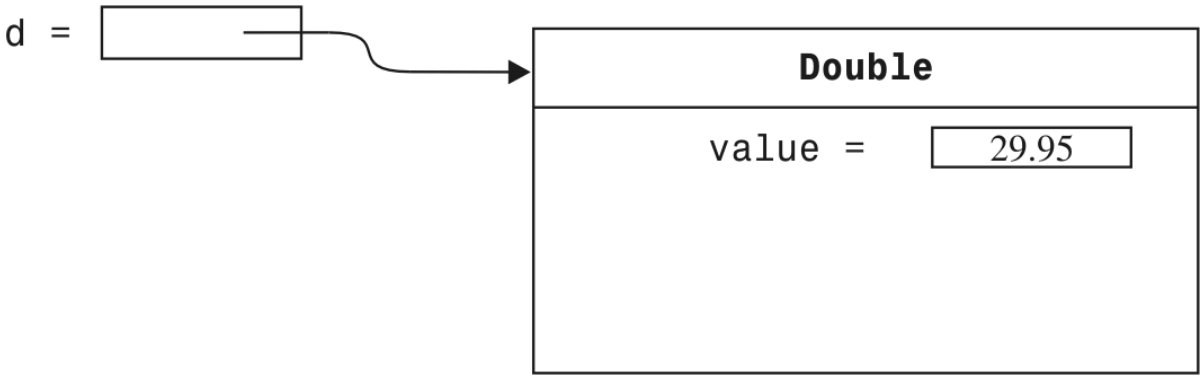
- Il metodo **toString** ritorna una stringa corrispondente alla **rappresentazione testuale** dell'oggetto
  - E' invocato **implicitamente** dal compilatore e può essere **ridefinito**
  - Se non ridefinito, di default ritorna una stringa formata da: nome classe + @ + indirizzo di memoria dell'oggetto in *esadecimale*

Wrappers

Wrappers

- La classe ArrayList non permette di avere valori di tipo **primitivo**
- Per manipolare valori di tipo primitivo si usano le *"classi involucro"* o **wrappers**

```
ArrayList<Double> d = new ArrayList<Double>();
d.add(29.95);
double x = d.get(0);
```



Wrappers

- Esiste una classe wrapper per ogni tipo primitivo

Tipo primitivo	Classe involucro
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Boxing e Unboxing

- Da Java 5.0 in poi **conversione implicita** tra tipo primitivo ↔ wrapper corrispondente

```
Double d = 29.95; // auto-boxing
     // come se fosse Double d = new Double(29.95);

double x = d; // auto-unboxing
     // come se fosse double x = Double.valueOf(d);
```

Boxing e Unboxing

- Le conversioni automatiche funzionano per le espressioni aritmetiche. Ad es. se d è Double questa espressione è valida:

```
Double e = d + 1;
```

  - **`d`** da **`Double`** viene convertito in **`double`** (**unboxing**)
  - **`1`** viene convertito in **`double`**
  - Viene eseguita la somma e "*impacchettato*" il risultato in un nuovo wrapper di tipo **`Double`** (**boxing**)
  - Viene memorizzato in **`e`** il **riferimento** al wrapper appena creato