Programmazione II

A.A. 2022-23
Prof. Maria Tortorella

Array ed ArrayList

- Collezioni di oggetti: array e ArrayList
- Il concetti di wrapper classes, auto-boxing e `for` loop generalizzato
- Algoritmi fondamentali su array

Arrays

- Array: una sequenza di valori tutti dello stesso tipo
- Costruire un array:

```
new double[10]
```

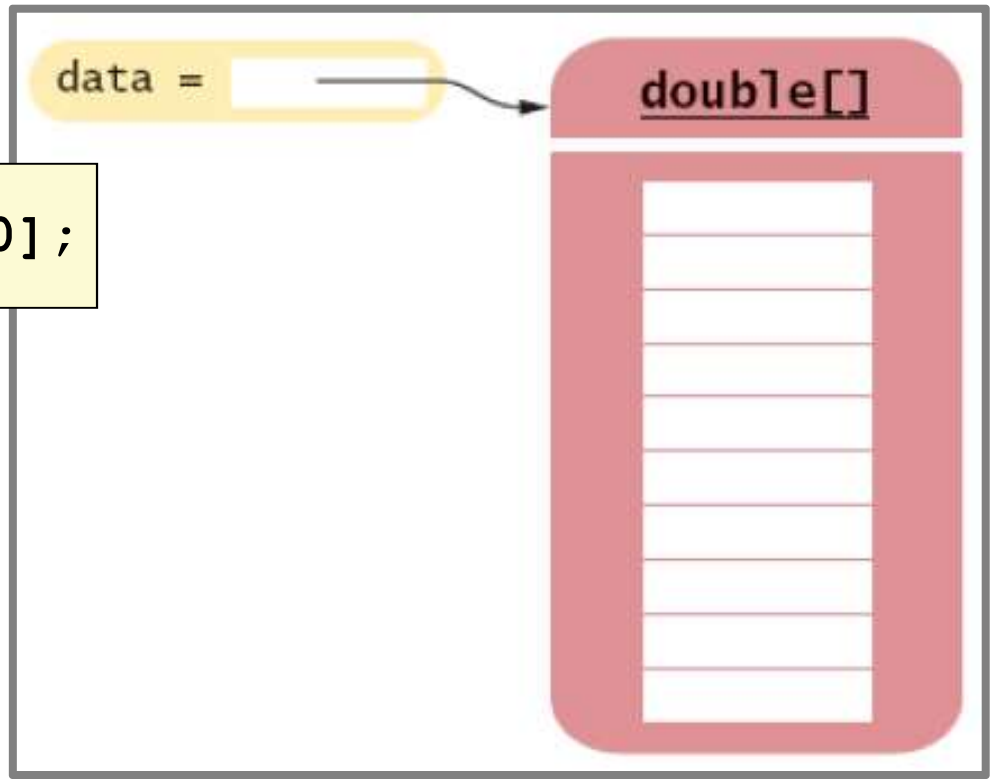
- Dichiarare e costruire un array di tipo `double[ ]`

```
double[] data = new double[10];
```

Arrays

- All'atto della creazione tutti i valori sono inizializzati nel seguente modo:
  - Numeri: 0
  - Boolean: `false`
  - Referenze ad oggetti: `null`

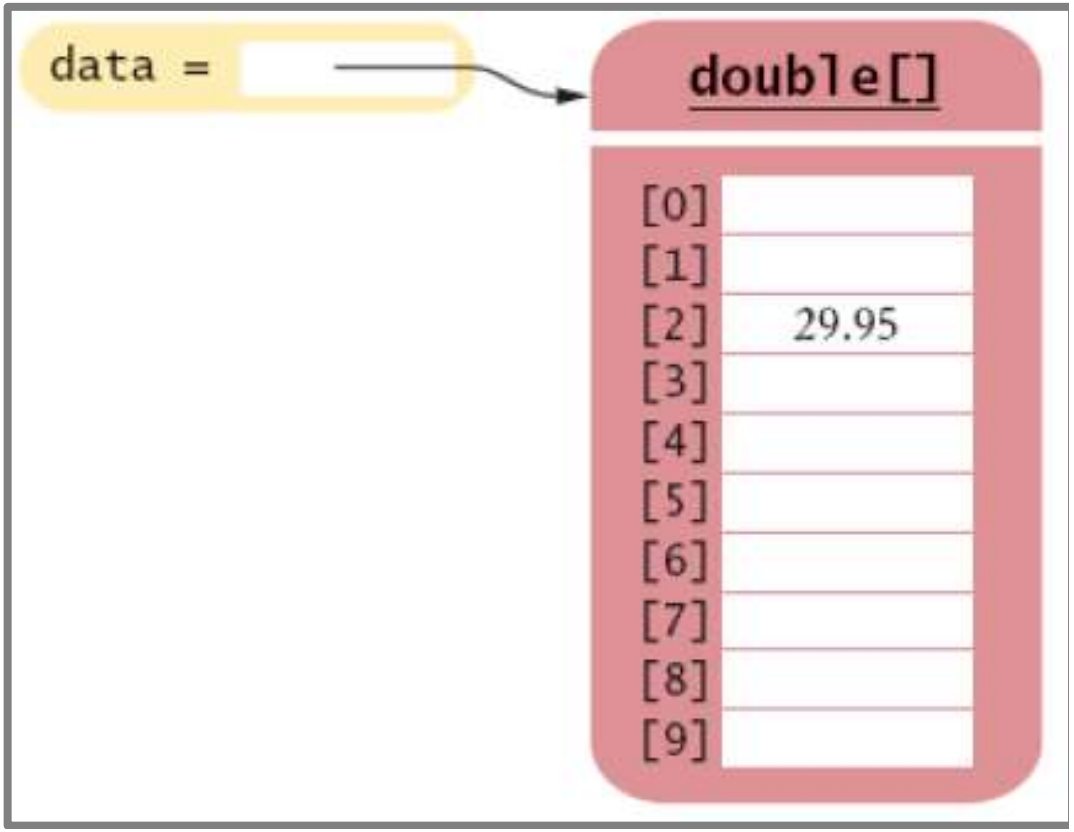
```
double[] data = new double[10];
```



Arrays

- Le parentesi quadrate [ ] consentono l'accesso agli elementi

```
data[2] = 29.95;
```



Arrays

- Un esempio d'uso del valore memorizzato:

```
System.out.println("The value of this item is " + data[4]);
```

- E' possibile conoscere la lunghezza di un array mediante: `data.length`
  - Non si tratta di un metodo!
- I valori dell'indice vanno da `0` a `length - 1`

Arrays

- Il tentativo di accedere un elemento oltre i limiti provoca un errore

```
double[] data = new double[10];
data[10] = 29.95; // ERROR
```

- Un limite fondamentale: la dimensione è fissa

Nota sintattica

Sintassi

`new` *`typeName`*`[`*`length`*`]`

Semantica

Costruisce un array con un dato numero di elementi

Esempio:

```
new double[10]
```

Sintassi

***`arrayReference`*`[`*`index`*`]`**

Semantica

Accede un elemento in una data posizione

Esempio:

```
data[2]
```

ArrayLists

Un altro tipo di collezione di oggetto

- La classe `ArrayList` gestisce sequenze di oggetti
- La dimensione varia nel tempo
- `ArrayList` fornisce metodi per moltissime operazioni comuni

- Prima aggiungere oggetti ad un ArrayList, è necessario creare un oggetto di tipo ArrayList:

```
ArrayList prova = new ArrayList ();
```

- È possibile aggiungere oggetti di tipo diverso – Da evitare

Array Lists

- La classe `ArrayList` è una classe generica
  - Può includere oggetti diversi … da evitare
- `ArrayList<T>` colleziona 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));
```

- Il metodo `size` restituisce la dimensione in un dato istante
  - Il numero di oggetti della collezione

Leggere un elemento

- Ciascun oggetto incluso nell'ArrayList ha una posizione
- Come per gli array il numero delle posizioni parte da 0
- Il metodo get restituisce l'oggetto di una data posizione

```
BankAccount anAccount = accounts.get(2);
   // gets the third element of the array list
```

- Un "bound error" segnala l'uso di un indice fuori range

```
int i = accounts.size();
anAccount = accounts.get(i); // Error
// legal index values are 0. . .i-1
```

Aggiungere un elemento

- Il metodo `set` sovrascrive un elemento

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

- Il metodo `add` aggiunge il nuovo valore

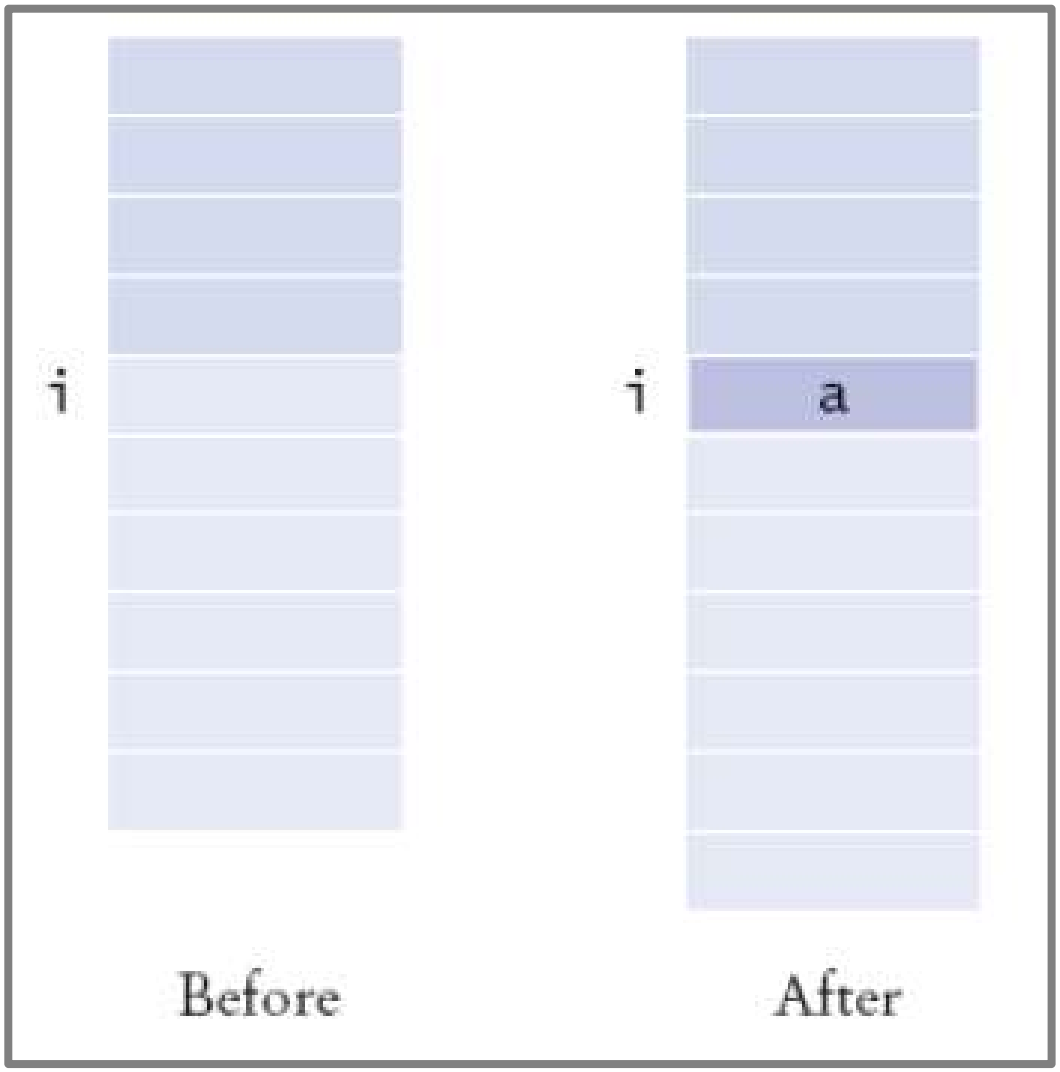
```
accounts.add(anAccount)
```

- Lo aggiunge all'indice dato se indicata una posizione. Tutti gli altri valori vengono shiftati in avanti di una posizione

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

Aggiungere un elemento

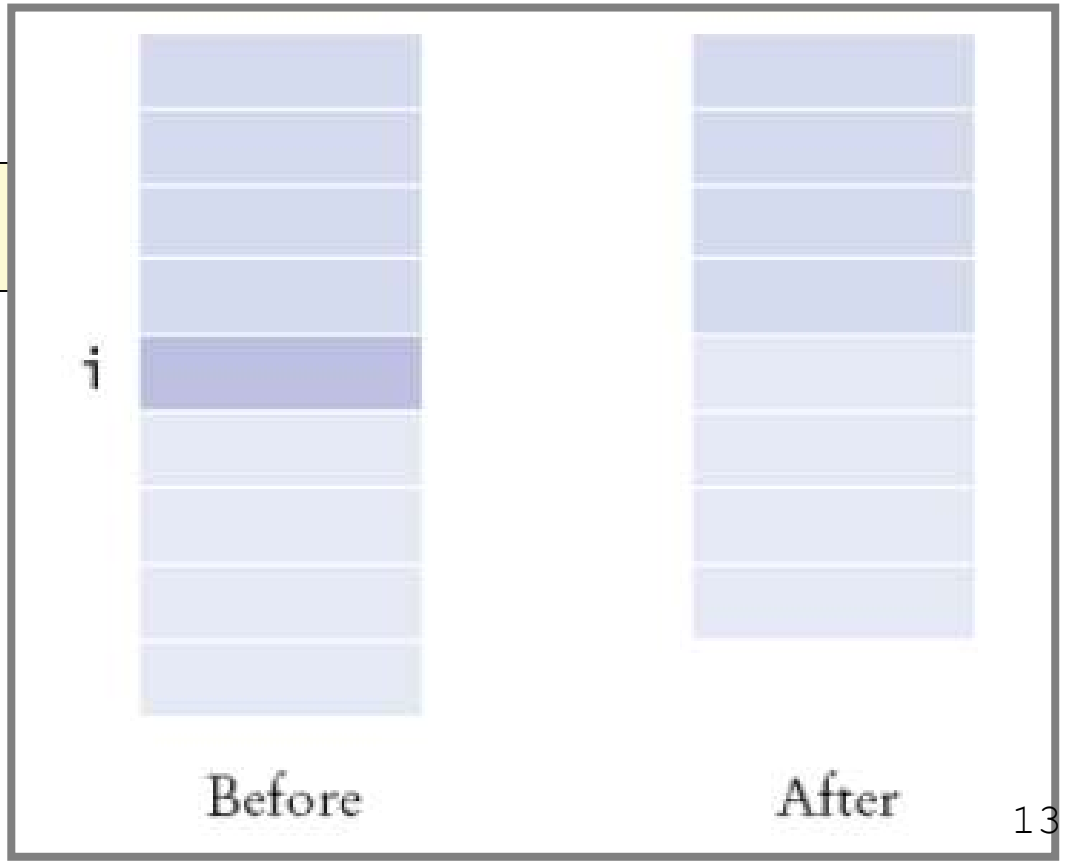
```
accounts.add(i, a);
```



Rimuovere un elemento

- Il metodo `remove` rimuove l'elemento all'indice dato. Gli elementi successivi sono shiftati di una posizione indietro

```
accounts.remove(i);
```



Un esempio

ArrayListTester.java

```
01: import java.util.ArrayList;
02:
03: /**
04:    This program tests the ArrayList class.
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);
```

File: 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 =
23:                         accounts.get(accounts.size() - 1);
24:         System.out.println("last account number="
25:                         + last.getAccountNumber());
26:     }
27: }
```

File: BankAccount.java

```
01: /**
02:    A bank account has a balance that can be changed by
03:    deposits and withdrawals.
04: */
05: public class BankAccount
06: {
07:    /**
08:       Constructs a bank account with a zero balance
09:       @param anAccountNumber the account number for
10:       this account */
11:    public BankAccount(int anAccountNumber)
12:    {
13:       accountNumber = anAccountNumber;
14:       balance = 0;
15:    }
16:
```

File: BankAccount.java

```
17: /**
18: Constructs a bank account with a given balance
19: @param anAccountNumber the account number for this accou
20: @param initialBalance the initial balance
21: */
22:    public BankAccount(int anAccNum, double initBalance)
23:    {
24:       accountNumber = anAccNum;
25:       balance = initBalance;
26:    }
27:
28:    /**
29:       Gets the account number of this bank account.
30:       @return the account number
31:    */
32:    public int getAccountNumber()
33:    {
34:       return accountNumber;
35:    }
```

File: BankAccount.java

```
36:
37:    /**
38:       Deposits money into the bank account.
39:       @param amount the amount to deposit
40:    */
41:    public void deposit(double amount)
42:    {
43:       double newBalance = balance + amount;
44:       balance = newBalance;
45:    }
46:
47:    /**
48:       Withdraws money from the bank account.
49:       @param amount the amount to withdraw
50:    */
51:    public void withdraw(double amount)
52:    {
53:       double newBalance = balance - amount;
54:       balance = newBalance;
```

File: BankAccount.java

```
55:     }
56:
57:     /**
58:        Gets the current balance of the bank account.
59:        @return the current balance
60:     */
61:     public double getBalance()
62:     {
63:        return balance;
64:     }
65:
66:     private int accountNumber;
67:     private double balance;
68: }
```

Output

```
size=3
first account number=1008
last account number=1729
```

Wrappers

- Non è possibile inserire tipi primitivi direttamente in un ArrayLists
- A tal fine si utilizzano le classi wrapper:

```
ArrayList<Double> data = new ArrayList<Double>();
data.add(29.95);                  // Autoboxing
double x = data.get(0);           // AutoUnboxing
```

Il ciclo `for` generalizzato

- Per visitare tutti gli elementi di una collezione:

```
double[] data = . . .;
double sum = 0;
for (int i = 0; i < data.length; i++)
{
   double e = data[i];
   sum = sum + e;
}
```

```
double[] data = . . .;
double sum = 0;
for (double e : data) // You should read this loop as
{                     // "for each e in data"
   sum = sum + e;
}
```

Il ciclo `for` generalizzato

- Funziona anche con `ArrayLists` :

```
double sum = 0;
for (int i = 0; i < accounts.size(); i++)
{
   BankAccount a = accounts.get(i);
   sum = sum + a.getBalance();
}
```

```
ArrayList<BankAccount> accounts = . . . ;
double sum = 0;
for (BankAccount a : accounts)
{
   sum = sum + a.getBalance();
}
```

Nota sintattica

```
for (Type variable : collection)
    statement
```

Esempio:

```
for (double e : data)
    sum = sum + e;
```

Esegue un ciclo per ogni elemento in una collezione

Algoritmi: Contare i matches

- Verifica tutti gli elementi e conta gli elementi che rispettano una condizione data

```
public class Bank
{
   public int count(double atLeast)
   {
      int matches = 0;
      for (BankAccount a : accounts)
      {
         if (a.getBalance() >= atLeast) // Found a match
            matches++;
      }
      return matches;
   }
   . . .
   private ArrayList<BankAccount> accounts;
}
```

Algoritmi: cercare un valore

- Cercare un valore che rispetta una data condizione.

```
public class Bank {
   public BankAccount find(int accountNumber) {
      boolean trovato = false;
      BankAccount a;
      int i = 0;
      while (i < accounts.size() && !trovato) {
        a = accounts.get(i);
        if (a.getAccountNumber() == accountNumber)
            trovato = true;   // Found a match
        i++;
      }
      if (trovato) return a;
      return null; // No match in the entire array list
   }
   . . .
}
```

Algoritmi: cercare un valore

- Cercare un valore che rispetta una data condizione

```
public class Bank
{
   public BankAccount find(int accountNumber)
   {
      for (BankAccount a : accounts)
      {
         if (a.getAccountNumber() == accountNumber)
                                              // Found a match
            return a;
      }
      return null; // No match in the entire array list
   }
   . . .
}
```

Algoritmi: cercare massimo

- Inizializzare il candidato con il primo valore della collezione
- Confrontare il candidato con tutti gli altri elementi
- Aggiornarlo se si trova un elemento più grande

```
public BankAccount getMaximum(){
      if (accounts.size() == 0) return null;
      BankAccount largestYet = accounts.get(0);
      for (int i = 1; i < accounts.size(); i++) {
            BankAccount a = accounts.get(i);
            if (a.getBalance() > largestYet.getBalance())
            largestYet = a;
      }
      return largestYet;
}
```

Analogamente per la ricerca del minimo

Algoritmi: cercare massimo

- Se la collezione non ha elementi si restituisce null

```
if (accounts.size() == 0) return null;
BankAccount largestYet = accounts.get(0);
. . .
```

Esempio: gestore di una banca

- Completare la classe Bank

```
// costruttore
public Bank()
// aggiunge un conto
public void addAccount(BankAccount a)
// totale su tutti i contiaperti
public double getTotalBalance()
// numero di conti aperti che abbiano un importo minimo
  dato
public int count(double atLeast)
// cerca un conto corrente
public BankAccount find(int accountNumber)
// restituisce il conto con la disponibilità massima
public BankAccount getMaximum()
```

Copia

- Copiare il riferimento ad un array NON comporta la copia dell'array

```
double[] data = new double[10];
// fill array . . .
double[] prices = data;
```