Capitolo 7

Implementazione delle classi

Sommario: Implementazione delle classi

Classi

Una classe è una ricetta che descrive come sono fatti i suoi oggetti.

Classi

Una classe è una ricetta che descrive come sono fatti i suoi oggetti.

In particolare definisce:

- Campi
  I dati che costituiscono lo stato di ogni singolo oggetto

Classi

Una classe è una ricetta che descrive come sono fatti i suoi oggetti.

In particolare definisce:

- Campi
  I dati che costituiscono lo stato di ogni singolo oggetto
- Costruttori
  Le porzioni di codice che inizializzano in modo opportuno lo stato di ogni nuovo oggetto

Classi

Una classe è una ricetta che descrive come sono fatti i suoi oggetti.

In particolare definisce:

- Campi
  I dati che costituiscono lo stato di ogni singolo oggetto
- Costruttori
  Le porzioni di codice che inizializzano in modo opportuno lo stato di ogni nuovo oggetto
- Metodi
  I comportamenti di ogni singolo oggetto

Classi

Una classe è una ricetta che descrive come sono fatti i suoi oggetti.

In particolare definisce:

- Campi
  I dati che costituiscono lo stato di ogni singolo oggetto
- Costruttori
  Le porzioni di codice che inizializzano in modo opportuno lo stato di ogni nuovo oggetto
- Metodi
  I comportamenti di ogni singolo oggetto

La classe dev'essere progettata in modo indipendente dalle applicazioni che la utilizzeranno

La classe Frazione

```
public class Frazione {
  //CAMPI
  ...

  //COSTRUTTORI
  ...

  //METODI
  ...
}
```

Frazione: i campi

Una frazione è determinata da due numeri interi, il numeratore e il denominatore

Frazione: i campi

Una frazione è determinata da due numeri interi, il numeratore e il denominatore

```
public class Frazione {
  //CAMPI
  private int num;  //il numeratore
  private int den;  //il denominatore

  //COSTRUTTORI
  ...
  //METODI
  ...
}
```

Frazione: i campi

Una frazione è determinata da due numeri interi, il numeratore e il denominatore

```
public class Frazione {
  //CAMPI
  private int num;  //il numeratore
  private int den;  //il denominatore

  //COSTRUTTORI
  ...
  //METODI
  ...
}
```

- La rappresentazione dell'oggetto riguarda solo chi progetta la classe, non chi la utilizza
- Nascondiamo questi campi dichiarandoli `private` (privati al codice della classe)

Frazione: costruttori

Quando viene costruito un oggetto:

```
new Frazione(3,4)
```

Frazione: costruttori

Quando viene costruito un oggetto:

```
new Frazione(3,4)
```

(1) viene riservato lo spazio in memoria per l'oggetto

- informazioni di controllo: il nome della classe a partire dalla quale è stato costruito

Frazione: costruttori

Quando viene costruito un oggetto:

```
new Frazione(3,4)
```

(1) viene riservato lo spazio in memoria per l'oggetto

- informazioni di controllo: il nome della classe a partire dalla quale è stato costruito
- stato: lo spazio per memorizzare i campi definiti nella classe

Frazione: costruttori

Quando viene costruito un oggetto:

```
new Frazione(3,4)
```

(1) viene riservato lo spazio in memoria per l'oggetto

- informazioni di controllo: il nome della classe a partire dalla quale è stato costruito
- stato: lo spazio per memorizzare i campi definiti nella classe

(2) vengono eseguite le istruzioni scritte nel codice del costruttore

Frazione: costruttori

Quando viene costruito un oggetto:

```
new Frazione(3,4)
```

(1) viene riservato lo spazio in memoria per l'oggetto

- informazioni di controllo: il nome della classe a partire dalla quale è stato costruito
- stato: lo spazio per memorizzare i campi definiti nella classe

(2) vengono eseguite le istruzioni scritte nel codice del costruttore

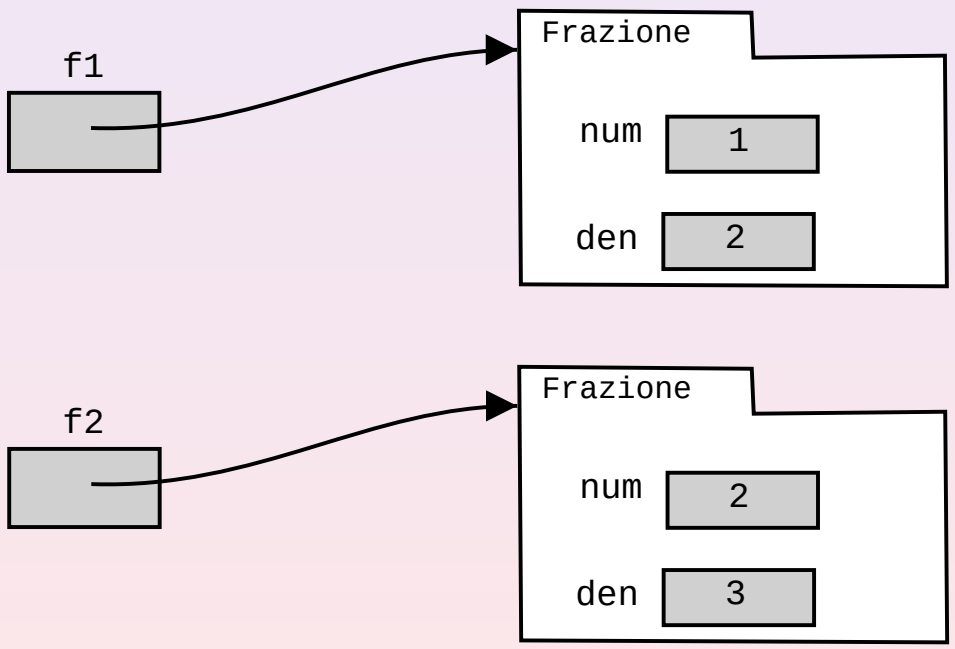
Nel nostro caso il costruttore deve assegnare ai campi i valori forniti come argomento

Frazione: costruttori

```
public class Frazione {
  //CAMPI
  private int num;  //il numeratore
  private int den;  //il denominatore

  //COSTRUTTORI
  public Frazione(int x, int y) {
    num = x;
    den = y;
  }

  ...

  //METODI
  ...
}
```

Costruzione degli oggetti

```
Frazione f1 = new Frazione(1,2);
Frazione f2 = new Frazione(2,3);
```



Frazione: costruttori

```
public class Frazione {
  //CAMPI
  private int num;  //il numeratore
  private int den;  //il denominatore

  //COSTRUTTORI
  public Frazione(int x, int y) {
    num = x;
    den = y;
  }

  public Frazione(int x) {
    num = x;
    den = 1;
  }

  //METODI
  ...
}
```

Frazione: il metodo per

- `public Frazione per(Frazione f)`
  Restituisce il riferimento a un nuovo oggetto che rappresenta la frazione ottenuta moltiplicando la frazione specificata come argomento con quella che esegue il metodo.

Frazione: il metodo per

- `public Frazione per(Frazione f)`
  Restituisce il riferimento a un nuovo oggetto che rappresenta la frazione ottenuta moltiplicando la frazione specificata come argomento con quella che esegue il metodo.

Schema

calcola il numeratore della nuova frazione
calcola il denominatore della nuova frazione
costruisci la nuova frazione
restituisci un riferimento a essa

Accedere ai campi di un oggetto

Per accedere a un campo

riferimento_oggetto.*nome_campo*

Accedere ai campi di un oggetto

Per accedere a un campo

riferimento_oggetto.*nome_campo*

- I campi di un oggetto sono accessibili all'interno della classe anche se dichiarati `private`

Accedere ai campi di un oggetto

Per accedere a un campo

riferimento_oggetto.*nome_campo*

- I campi di un oggetto sono accessibili all'interno della classe anche se dichiarati `private`
- Riferimento `this`:
  - in un metodo si riferisce all'oggetto che esegue il metodo

Accedere ai campi di un oggetto

Per accedere a un campo

riferimento_oggetto.*nome_campo*

- I campi di un oggetto sono accessibili all'interno della classe anche se dichiarati `private`
- Riferimento `this`:
  - in un metodo si riferisce all'oggetto che esegue il metodo
  - in un costruttore si riferisce all'oggetto che si sta costruendo

Esempio

```
public class Frazione {
  //CAMPI
  private int num;  //il numeratore
  private int den;  //il denominatore

  //COSTRUTTORI
  public Frazione(int x, int y) {
    this.num = x;
    this.den = y;
  }
  ...
```

Frazione: il metodo per

```
public class Frazione {
  //CAMPI
  private int num;  //il numeratore
  private int den;  //il denominatore
  ...
  //METODI
  ...
  public Frazione per(Frazione f) {
    int n = this.num * f.num;
    int d = this.den * f.den;
    Frazione g = new Frazione(n, d);
    return g;
  }
  ...
}
```

`return` è la parola chiave che indica la restituzione del valore al chiamante, l'argomento dev'essere un'espressione del tipo restituito dal metodo.

Sommario: Implementazione delle classi

Variabili locali

Sono le variabili definite nel corpo di un metodo o di un costruttore.

Variabili locali

Sono le variabili definite nel corpo di un metodo o di un costruttore.

Esempio

```
public Frazione per(Frazione f) {
  int n = this.num * f.num;
  int d = this.den * f.den;
  Frazione g = new Frazione(n, d);
  return g;
}
```

Variabili locali

Sono le variabili definite nel corpo di un metodo o di un costruttore.

Esempio

```
public Frazione per(Frazione f) {
  int n = this.num * f.num;
  int d = this.den * f.den;
  Frazione g = new Frazione(n, d);
  return g;
}
```

- Sono utilizzate per la memorizzazione temporanea di valori durante l'esecuzione del metodo

Variabili locali

Sono le variabili definite nel corpo di un metodo o di un costruttore.

Esempio

```
public Frazione per(Frazione f) {
  int n = this.num * f.num;
  int d = this.den * f.den;
  Frazione g = new Frazione(n, d);
  return g;
}
```

- Sono utilizzate per la memorizzazione temporanea di valori durante l'esecuzione del metodo
- Quando l'esecuzione del metodo termina vengono distrutte

Variabili locali

Sono le variabili definite nel corpo di un metodo o di un costruttore.

Esempio

```
public Frazione per(Frazione f) {
  int n = this.num * f.num;
  int d = this.den * f.den;
  Frazione g = new Frazione(n, d);
  return g;
}
```

- Sono utilizzate per la memorizzazione temporanea di valori durante l'esecuzione del metodo
- Quando l'esecuzione del metodo termina vengono distrutte
- Non vengono inizializzate automaticamente

Parametri formali

Sono le variabili dichiarate nell'intestazione di un metodo o di un costruttore.

Parametri formali

Sono le variabili dichiarate nell'intestazione di un metodo o di un costruttore.

Esempio

```
public Frazione(int x, int y){
...
}
```

Parametri formali

Sono le variabili dichiarate nell'intestazione di un metodo o di un costruttore.

Esempio

```
public Frazione(int x, int y){
...
}
```

- Sono utilizzati per comunicare le informazioni necessarie all'esecuzione del metodo/costruttore

Parametri formali

Sono le variabili dichiarate nell'intestazione di un metodo o di un costruttore.

Esempio

```
public Frazione(int x, int y){
...
}
```

- Sono utilizzati per comunicare le informazioni necessarie all'esecuzione del metodo/costruttore
- Sono variabili utilizzabili all'interno del codice del metodo/costruttore

Parametri formali

Sono le variabili dichiarate nell'intestazione di un metodo o di un costruttore.

Esempio

```
public Frazione(int x, int y){
...
}
```

- Sono utilizzati per comunicare le informazioni necessarie all'esecuzione del metodo/costruttore
- Sono variabili utilizzabili all'interno del codice del metodo/costruttore
- Vengono inizializzati al momento dell'invocazione con il valore degli argomenti (parametri attuali)

Parametri formali

Sono le variabili dichiarate nell'intestazione di un metodo o di un costruttore.

Esempio

```
public Frazione(int x, int y){
...
}
```

- Sono utilizzati per comunicare le informazioni necessarie all'esecuzione del metodo/costruttore
- Sono variabili utilizzabili all'interno del codice del metodo/costruttore
- Vengono inizializzati al momento dell'invocazione con il valore degli argomenti (parametri attuali)
- Quando l'esecuzione del metodo termina vengono distrutti

Parametri attuali

Sono le espressioni specificate all'atto dell'invocazione del metodo o del costruttore.

Parametri attuali

Sono le espressioni specificate all'atto dell'invocazione del metodo o del costruttore.

Esempi

- `public Frazione(int x, int y)`

```
Frazione f = new Frazione(3 * x, 2);
```

Parametri attuali

Sono le espressioni specificate all'atto dell'invocazione del metodo o del costruttore.

Esempi

- `public Frazione(int x, int y)`

```
Frazione f = new Frazione(3 * x, 2);
```

- `public Frazione per(Frazione f)`

```
f.per((new Frazione(3, 5)).per(f))
```

Parametri attuali

Sono le espressioni specificate all'atto dell'invocazione del metodo o del costruttore.

Esempi

- `public Frazione(int x, int y)`

```
Frazione f = new Frazione(3 * x, 2);
```

- `public Frazione per(Frazione f)`

```
f.per((new Frazione(3, 5)).per(f))
```

- Devono essere di un tipo compatibile con quello del parametro formale corrispondente

Parametri attuali

Sono le espressioni specificate all'atto dell'invocazione del metodo o del costruttore.

Esempi

- `public Frazione(int x, int y)`

```
Frazione f = new Frazione(3 * x, 2);
```

- `public Frazione per(Frazione f)`

```
f.per((new Frazione(3, 5)).per(f))
```

- Devono essere di un tipo compatibile con quello del parametro formale corrispondente
- All'atto dell'esecuzione vengono valutati e il loro valore valore viene utilizzato per inizializzare i parametri formali

Frazione: i metodi piu e meno

```
public Frazione piu(Frazione f) {
  int n = this.num * f.den + this.den * f.num;
  int d = this.den * f.den;
  return new Frazione(n, d);
}
```

Frazione: i metodi piu e meno

```
public Frazione piu(Frazione f) {
  int n = this.num * f.den + this.den * f.num;
  int d = this.den * f.den;
  return new Frazione(n, d);
}

public Frazione meno(Frazione f) {
  int n = this.num * f.den - this.den * f.num;
  int d = this.den * f.den;
  return new Frazione(n, d);
}
```

Frazione: il metodo equals

- `public boolean equals(Frazione f)`
  Confronta la frazione rappresentata dall'oggetto che esegue il metodo con la frazione specificata come argomento. Restituisce `true` se le due frazioni rappresentano lo stesso valore e `false` altrimenti.

Frazione: il metodo equals

- `public boolean equals(Frazione f)`
  Confronta la frazione rappresentata dall'oggetto che esegue il metodo con la frazione specificata come argomento. Restituisce `true` se le due frazioni rappresentano lo stesso valore e `false` altrimenti.

Schema

calcola la differenza tra le due frazioni
SE il numeratore della frazione risultante vale zero
  ALLORA
    restituisci il valore `true`
  ALTRIMENTI
    restituisci il valore `false`
FINESE

Frazione: il metodo equals

```
public boolean equals(Frazione f) {
  Frazione g = this.meno(f);
  if (g.num == 0)
    return true;
  else
    return false;
}
```

Frazione: il metodo equals

```
public boolean equals(Frazione f) {
  Frazione g = this.meno(f);
  if (g.num == 0)
    return true;
  else
    return false;
}
```

Istruzione `return`:

- seguita da un'espressione di un tipo compatibile con il tipo restituito dal metodo

Frazione: il metodo equals

```
public boolean equals(Frazione f) {
  Frazione g = this.meno(f);
  if (g.num == 0)
    return true;
  else
    return false;
}
```

Istruzione `return`:

- seguita da un'espressione di un tipo compatibile con il tipo restituito dal metodo
- ogni possibile percorso di esecuzione del metodo deve contenere un'istruzione `return`

Frazione: il metodo equals

```
public boolean equals(Frazione f) {
  Frazione g = this.meno(f);
  if (g.num == 0)
    return true;
  else
    return false;
}
```

Istruzione `return`:

- seguita da un'espressione di un tipo compatibile con il tipo restituito dal metodo
- ogni possibile percorso di esecuzione del metodo deve contenere un'istruzione `return`
- all'atto dell'esecuzione l'espressione viene valutata e il suo valore è il valore restituito dal metodo

Frazione: i metodi isMinore e isMaggiore

```
public boolean isMinore(Frazione f) {
  Frazione g = this.meno(f);
  if ((g.num < 0 && g.den > 0) || (g.num > 0 && g.den < 0))
    return true;
  else
    return false;
}
```

Frazione: i metodi isMinore e isMaggiore

```
public boolean isMinore(Frazione f) {
  Frazione g = this.meno(f);
  if ((g.num < 0 && g.den > 0) || (g.num > 0 && g.den < 0))
    return true;
  else
    return false;
}

public boolean isMaggiore(Frazione f) {
  Frazione g = this.meno(f);
  if ((g.num < 0 && g.den < 0) || (g.num > 0 && g.den > 0))
    return true;
  else
    return false;
}
```

Frazione: il metodo toString

```
public String toString() {
  return num + "/" + den;
}
```

Frazione: il metodo toString

```
public String toString() {
  return num + "/" + den;
}
```

Osservazione

`this` può essere omesso, sottintendendo così che ci riferiamo ai campi dell'oggetto stesso.

Frazione: il metodo toString

```
public String toString() {
  return num + "/" + den;
}
```

Problema

Se il denominatore contiene zero, la frazione non ha senso.

Frazione: il metodo toString

```
public String toString() {
  return num + "/" + den;
}
```

Problema

Se il denominatore contiene zero, la frazione non ha senso.

```
public String toString() {
  if (den == 0)
    return "impossibile";
  else
    return num + "/" + den;
}
```

Frazione: miglioramenti - costruttori

```
public Frazione(int x, int y) {
  num = x;
  den = y;
}

public Frazione(int x) {
  num = x;
  den = 1;
}
```

Osservazione

Il secondo costruttore può essere simulato dal primo fornendo 1 come valore del denominatore.

this per invocare un costruttore

```
public Frazione(int x, int y) {
  num = x;
  den = y;
}

public Frazione(int x) {
  this(x, 1);
}
```

this per invocare un costruttore

```
public Frazione(int x, int y) {
  num = x;
  den = y;
}

public Frazione(int x) {
  this(x, 1);
}
```

- Nel corpo del costruttore è possibile richiamare un altro costruttore della stessa classe utilizzando `this` seguito dalla lista degli argomenti

this per invocare un costruttore

```
public Frazione(int x, int y) {
  num = x;
  den = y;
}

public Frazione(int x) {
  this(x, 1);
}
```

- Nel corpo del costruttore è possibile richiamare un altro costruttore della stessa classe utilizzando `this` seguito dalla lista degli argomenti
- L'invocazione del costruttore tramite `this` dev'essere la prima istruzione del codice del costruttore

`Frazione`: miglioramenti - `diviso`

Osservazione

Per come è definita `Frazione` è possibile costruire oggetti che non hanno senso dal punto di vista matematico (con denominatore uguale a zero).

Frazione: miglioramenti - diviso

Osservazione

Per come è definita `Frazione` è possibile costruire oggetti che non hanno senso dal punto di vista matematico (con denominatore uguale a zero).

Alcuni metodi trattano in modo consistente le frazioni *impossibili*:

Frazione: miglioramenti - diviso

Osservazione

Per come è definita `Frazione` è possibile costruire oggetti che non hanno senso dal punto di vista matematico (con denominatore uguale a zero).

Alcuni metodi trattano in modo consistente le frazioni *impossibili*:

- `toString`: restituisce `"impossibile"` per tali oggetti

Frazione: miglioramenti - diviso

Osservazione

Per come è definita `Frazione` è possibile costruire oggetti che non hanno senso dal punto di vista matematico (con denominatore uguale a zero).

Alcuni metodi trattano in modo consistente le frazioni *impossibili*:

- `toString`: restituisce `"impossibile"` per tali oggetti
- `piu`, `meno`, `per`: se uno degli oggetti coinvolti rappresenta una frazione `"impossibile"` restituiscono il riferimento a una frazione `"impossibile"`

```
public Frazione piu(Frazione f) {
  int n = this.num * f.den + this.den * f.num;
  int d = this.den * f.den;
  return new Frazione(n, d);
}
```

Frazione: miglioramenti - diviso

```
public Frazione diviso(Frazione f) {
  int n = this.num * f.den;
  int d = this.den * f.num;
  return new Frazione(n, d);
}
```

Scorretto se `f` è una frazione possibile e `this` è impossibile

Frazione: miglioramenti - diviso

```
public Frazione diviso(Frazione f) {
  int n = this.num * f.den;
  int d = this.den * f.num;
  return new Frazione(n, d);
}
```

Scorretto se `f` è una frazione possibile e `this` è impossibile

```
public Frazione diviso(Frazione f) {
  if (f.den == 0)
    return new Frazione(0, 0);
  else {
    int n = this.num * f.den;
    int d = this.den * f.num;
    return new Frazione(n, d);
  }
}
```

Frazione: miglioramenti - toString

Se il denominatore vale 1 vogliamo che sia restituita una stringa in cui non compare l'indicazione del denominatore.

Frazione: miglioramenti - toString

Se il denominatore vale 1 vogliamo che sia restituita una stringa in cui non compare l'indicazione del denominatore.

Schema

```
SE il denominatore è uguale a 0
  ALLORA
    restituisci "impossibile"
  ALTRIMENTI
    SE il denominatore è uguale a 1
    ALLORA
      restituisci il valore del numeratore
    ALTRIMENTI
      restituisci la stringa della forma numeratore/denominatore
    FINESE
FINESE
```

Frazione: miglioramenti - toString

```
public String toString() {
  if (den == 0)
    return "impossibile";
  else if (den == 1)
    return String.valueOf(num);
  else
    return num + "/" + den;
}
```

Cambiamento della rappresentazione

Le frazioni vengono rappresentate senza tener conto delle semplificazioni.

Cambiamento della rappresentazione

Le frazioni vengono rappresentate senza tener conto delle semplificazioni.

- Sarebbe opportuno che il metodo `toString` fornisse `"1/4"` in luogo di `"32/128"`.

Cambiamento della rappresentazione

Le frazioni vengono rappresentate senza tener conto delle semplificazioni.

- Sarebbe opportuno che il metodo `toString` fornisse `"1/4"` in luogo di `"32/128"`.
- Scelte possibili:
  - semplificare le frazioni prima di visualizzarle
  - rappresentare frazioni semplificate, cioè semplificarle al momento della costruzione

Cambiamento della rappresentazione

Le frazioni vengono rappresentate senza tener conto delle semplificazioni.

- Sarebbe opportuno che il metodo `toString` fornisse `"1/4"` in luogo di `"32/128"`.
- Scelte possibili:
  - semplificare le frazioni prima di visualizzarle
  - rappresentare frazioni semplificate, cioè semplificarle al momento della costruzione
- Scegliamo la seconda soluzione

Il nuovo costruttore

Schema

```
public Frazione(int x, int y) {
  sia m il massimo comun divisore di x e y;
  num = x / m;
  den = y / m;
}
```

Il nuovo costruttore

Schema

```
public Frazione(int x, int y) {
  sia m il massimo comun divisore di x e y;
  num = x / m;
  den = y / m;
}
```

- Potremmo inserire il codice per il calcolo del mcd direttamente nel costruttore

Il nuovo costruttore

Schema

```
public Frazione(int x, int y) {
  sia m il massimo comun divisore di x e y;
  num = x / m;
  den = y / m;
}
```

- Potremmo inserire il codice per il calcolo del mcd direttamente nel costruttore
- Per modularizzare meglio il codice è comunque bene introdurre un metodo apposito

Il nuovo costruttore

Schema

```
public Frazione(int x, int y) {
  sia m il massimo comun divisore di x e y;
  num = x / m;
  den = y / m;
}
```

- Potremmo inserire il codice per il calcolo del mcd direttamente nel costruttore
- Per modularizzare meglio il codice è comunque bene introdurre un metodo apposito
  - Non svolge azioni legate a un singolo oggetto, ma esegue un calcolo utile alla classe per la costruzione di nuovi oggetti (statico).

Il nuovo costruttore

Schema

```
public Frazione(int x, int y) {
  sia m il massimo comun divisore di x e y;
  num = x / m;
  den = y / m;
}
```

- Potremmo inserire il codice per il calcolo del mcd direttamente nel costruttore
- Per modularizzare meglio il codice è comunque bene introdurre un metodo apposito
  - Non svolge azioni legate a un singolo oggetto, ma esegue un calcolo utile alla classe per la costruzione di nuovi oggetti (statico).
  - È un servizio utile all'interno che non deve essere visibile all'esterno (privato)

mcd: implementazione dell'algoritmo di Euclide

```
private static int mcd(int a, int b) {
  int resto;
  do {
    resto = a % b;
    a = b;
    b = resto;
  } while (resto != 0);
  return a;
}
```

mcd: implementazione dell'algoritmo di Euclide

```
private static int mcd(int a, int b) {
  int resto;
  do {
    resto = a % b;
    a = b;
    b = resto;
  } while (resto != 0);
  return a;
}
```

Il costruttore

```
  public Frazione(int x, int y) {
    int m = mcd(x,y);
    num = x / m;
    den = y / m;
  }
```

Il costruttore: rappresentazione del segno

Vogliamo che frazioni con lo stesso valore vengono memorizzate nello stesso modo.

Ad esempio:

```
2/-4      -3/6      -4/8
```

rappresentate con -1 nel campo `num` e 2 nel campo `den`.

Il costruttore: rappresentazione del segno

Vogliamo che frazioni con lo stesso valore vengono memorizzate nello stesso modo.

Ad esempio:

```
2/-4      -3/6      -4/8
```

rappresentate con -1 nel campo `num` e 2 nel campo `den`.

- In questo modo il metodo `equals` può essere realizzato verificando che i valori dei campi siano gli stessi

Il costruttore: rappresentazione del segno

Vogliamo che frazioni con lo stesso valore vengono memorizzate nello stesso modo.

Ad esempio:

```
2/-4     -3/6     -4/8
```

rappresentate con -1 nel campo `num` e 2 nel campo `den`.

- In questo modo il metodo `equals` può essere realizzato verificando che i valori dei campi siano gli stessi
- Riscriviamo il costruttore tenendo conto dei seguenti casi particolari:
  - uno degli argomenti è negativo
  - il secondo parametro è uguale a zero

Il costruttore

```
public Frazione(int x, int y) {
  if (y == 0) {
    num = 0;
    den = 0;
  } else {
    //memorizza il segno
    boolean negativo = (x < 0 && y > 0) || (x > 0 && y < 0);
    if (x < 0)
      x = - x;  //elimina l'eventuale segno meno davanti a x
    if (y < 0)
      y = - y;  //elimina l'eventuale segno meno davanti a y

    int m = mcd(x, y);
    if (negativo)
      num = - x / m;  //segno memorizzato al numeratore
    else
      num = x / m;
    den = y / m;
  }
}
```

Il metodo `diviso`

Se una frazione è impossibile sia `num` sia `den` contengono 0.

Il metodo `diviso`

Se una frazione è impossibile sia `num` sia `den` contengono 0.

```
public Frazione diviso(Frazione f) {
  int n = this.num * f.den;
  int d = this.den * f.num;
  return new Frazione(n, d);
}
```

Il metodo equals

```
public boolean equals(Frazione f) {
  if (this.num == f.num && this.den == f.den)
    return true;
  else
    return false;
}
```

Il metodo equals

```
public boolean equals(Frazione f) {
  if (this.num == f.num && this.den == f.den)
    return true;
  else
    return false;
}
```

Oppure

```
public boolean equals(Frazione f) {
  return this.num == f.num && this.den == f.den;
}
```

I metodi isMinore e isMaggiore

È sufficiente guardare il segno del numeratore della differenza.

```
public boolean isMinore(Frazione f) {
  Frazione g = this.meno(f);
  if (g.num < 0)
    return true;
  else
    return false;
}
```

I metodi isMinore e isMaggiore

È sufficiente guardare il segno del numeratore della differenza.

```
public boolean isMinore(Frazione f) {
  Frazione g = this.meno(f);
  if (g.num < 0)
    return true;
  else
    return false;
}

public boolean isMaggiore(Frazione f) {
  Frazione g = this.meno(f);
  if (g.num > 0)
    return true;
  else
    return false;
}
```

Frazione implementa Comparable<Frazione>

(1) Dichiariamo che `Frazione` implementa l'interfaccia generica `Comparable<T>` istanziando `T` con `Frazione`:

Frazione implementa Comparable<Frazione>

(1) Dichiariamo che `Frazione` implementa l'interfaccia generica `Comparable<T>` istanziando `T` con `Frazione`:

```
public class Frazione implements Comparable<Frazione> {
   ...
}
```

Frazione implementa Comparable<Frazione>

(1) Dichiariamo che `Frazione` implementa l'interfaccia generica `Comparable<T>` istanziando `T` con `Frazione`:

```
public class Frazione implements Comparable<Frazione> {
   ...
}
```

(2) Implementiamo il metodo:

- `public int compareTo(T o)`
  Confronta l'oggetto che esegue il metodo con quello specificato come argomento, e restituisce un intero negativo, zero, o un intero positivo, a seconda che l'oggetto che esegue il metodo sia minore, uguale o maggiore di quello specificato come argomento.

Il metodo compareTo(Frazione altra)

```
//IMPLEMENTAZIONE DI COMPARABLE
public int compareTo(Frazione altra) {
  if (this.equals(altra))
    return 0;
  else
    if (this.isMinore(altra))
      return -1;
    else
      return 1;
}
```

Specifica della classe `Orario`

Contratto

Le sue istanze rappresentano orari con la granularità dei minuti.

Specifica della classe `Orario`

Contratto

Le sue istanze rappresentano orari con la granularità dei minuti.

Costruttori

- `public Orario(int hh, int mm)`
  Costruisce un oggetto che rappresenta l'orario, in cui le ore sono date dal primo parametro e i minuti dal secondo.

Specifica della classe `Orario`

Contratto

Le sue istanze rappresentano orari con la granularità dei minuti.

Costruttori

- `public Orario(int hh, int mm)`
  Costruisce un oggetto che rappresenta l'orario, in cui le ore sono date dal primo parametro e i minuti dal secondo.

- `public Orario()`
  Costruisce un oggetto che rappresenta l'orario attuale, cioè l'orario relativo all'istante in cui viene invocato.

Specifica della classe `Orario`

Contratto

Le sue istanze rappresentano orari con la granularità dei minuti.

Costruttori

- `public Orario(int hh, int mm)`
  Costruisce un oggetto che rappresenta l'orario, in cui le ore sono date dal primo parametro e i minuti dal secondo.

- `public Orario()`
  Costruisce un oggetto che rappresenta l'orario attuale, cioè l'orario relativo all'istante in cui viene invocato.

- `public Orario(String s)`
  Costruisce un oggetto che rappresenta l'orario indicato nella stringa fornita tramite il parametro.

Metodi di accesso alle informazioni (metodi get)

- `public int getOre()`
  Restituisce il valore delle ore.

Metodi di accesso alle informazioni (metodi get)

- `public int getOre()`
  Restituisce il valore delle ore.
- `public int getMinuti()`
  Restituisce il valore dei minuti.

Metodi di confronto

- `public boolean equals(Orario altro)`
  Restituisce `true` quando l'orario rappresentato dall'oggetto che esegue il metodo coincide con quello rappresentato dall'oggetto fornito tramite il parametro.

Metodi di confronto

- `public boolean equals(Orario altro)`
  Restituisce `true` quando l'orario rappresentato dall'oggetto che esegue il metodo coincide con quello rappresentato dall'oggetto fornito tramite il parametro.

- `public boolean isMinore(Orario altro)`
  Restituisce `true` quando l'orario rappresentato dall'oggetto che esegue il metodo precede quello fornito tramite il parametro, supponendo che i due orari si riferiscano alla stessa giornata.

Metodi di confronto

- `public boolean equals(Orario altro)`
  Restituisce `true` quando l'orario rappresentato dall'oggetto che esegue il metodo coincide con quello rappresentato dall'oggetto fornito tramite il parametro.
- `public boolean isMinore(Orario altro)`
  Restituisce `true` quando l'orario rappresentato dall'oggetto che esegue il metodo precede quello fornito tramite il parametro, supponendo che i due orari si riferiscano alla stessa giornata.
- `public boolean isMaggiore(Orario altro)`
  Restituisce `true` quando l'orario rappresentato dall'oggetto che esegue il metodo segue quello fornito tramite il parametro, supponendo che i due orari si riferiscano alla stessa giornata.

Metodi di calcolo

- `public int quantoManca(Orario altro)`
  Restituisce il numero di minuti che intercorrono tra l'orario rappresentato dall'oggetto che esegue il metodo e quello rappresentato dall'oggetto fornito tramite il parametro, considerati come orari riferiti alla stessa giornata.

  Se l'orario rappresentato dall'oggetto che esegue il metodo è minore di quello fornito tramite il parametro, il risultato sarà un numero positivo; se invece è maggiore, il risultato sarà un numero negativo.

Metodi di calcolo

- `public int quantoManca(Orario altro)`
  Restituisce il numero di minuti che intercorrono tra l'orario rappresentato dall'oggetto che esegue il metodo e quello rappresentato dall'oggetto fornito tramite il parametro, considerati come orari riferiti alla stessa giornata.

  Se l'orario rappresentato dall'oggetto che esegue il metodo è minore di quello fornito tramite il parametro, il risultato sarà un numero positivo; se invece è maggiore, il risultato sarà un numero negativo.

- `public String toString()`
  Restituisce una stringa che descrive l'orario rappresentato dall'oggetto che esegue il metodo, come `"9:04"` o `"12:55"`.

Orario: struttura

```
//IMPORTAZIONI

class Orario {
  //CAMPI
  ...

  //COSTRUTTORI
  ...

  //METODI
  ...
}
```

Orario: campi e costruttori

```
//CAMPI
private int ore, min;
```

Orario: campi e costruttori

```
//CAMPI
private int ore, min;

//COSTRUTTORI
public Orario(int hh, int mm) {
  ore = hh;
  min = mm;
}
```

Orario: campi e costruttori

```
//CAMPI
private int ore, min;

//COSTRUTTORI
public Orario(int hh, int mm) {
  ore = hh;
  min = mm;
}

public Orario() {
  GregorianCalendar adesso = new GregorianCalendar();
  ore = adesso.get(Calendar.HOUR_OF_DAY);
  min = adesso.get(Calendar.MINUTE);
}
```

Orario: campi e costruttori

```
//CAMPI
private int ore, min;

//COSTRUTTORI
public Orario(int hh, int mm) {
  ore = hh;
  min = mm;
}

public Orario() {
  GregorianCalendar adesso = new GregorianCalendar();
  ore = adesso.get(Calendar.HOUR_OF_DAY);
  min = adesso.get(Calendar.MINUTE);
}

public Orario(String s) {
  ore = Integer.parseInt(s.substring(0,2));
  min = Integer.parseInt(s.substring(3,5));
}
```

Orario: metodi di accesso

```
public int getOre() {
  return ore;
}
```

Orario: metodi di accesso

```
public int getOre() {
  return ore;
}

public int getMinuti() {
  return min;
}
```

Orario: Metodi di confronto

```
public boolean equals(Orario altro) {
  return this.ore == altro.ore && this.min == altro.min;
}
```

Orario: Metodi di confronto

```
public boolean equals(Orario altro) {
  return this.ore == altro.ore && this.min == altro.min;
}

public boolean isMinore(Orario altro) {
  return ore < altro.ore ||
         (ore == altro.ore && min < altro.min);
}
```

Orario: Metodi di confronto

```
public boolean equals(Orario altro) {
  return this.ore == altro.ore && this.min == altro.min;
}

public boolean isMinore(Orario altro) {
  return ore < altro.ore ||
         (ore == altro.ore && min < altro.min);
}

public boolean isMaggiore(Orario altro) {
  return ore > altro.ore ||
         (ore == altro.ore && min > altro.min);
}
```

Orario: metodi di calcolo

```
public int quantoManca(Orario altro) {
  return (altro.ore - ore) * 60 + altro.min - min;
}
```

Orario: metodi di calcolo

```
public int quantoManca(Orario altro) {
  return (altro.ore - ore) * 60 + altro.min - min;
}

public String toString() {
  String stringaMinuti = (min < 10 ? "0" : "") + min;
  return ore + ":" + stringaMinuti;
}
```

Miglioramenti

(1) Possibilità di selezionare un separatore diverso da `':'` tra ore e minuti

Miglioramenti

(1) Possibilità di selezionare un separatore diverso da `':'` tra ore e minuti

Aggiungiamo un metodo

```
void setSeparatoreTo(char sep)
```

che modifichi il carattere utilizzato dal metodo `toString` per separare ore e minuti.

Miglioramenti

(1) Possibilità di selezionare un separatore diverso da `':'` tra ore e minuti

Aggiungiamo un metodo

```
void setSeparatoreTo(char sep)
```

che modifichi il carattere utilizzato dal metodo `toString` per separare ore e minuti.

(2) Possibilità di indicare gli orari su 12 ore invece che su 24 (`5:22pm`)

Miglioramenti

(1) Possibilità di selezionare un separatore diverso da `':'` tra ore e minuti

Aggiungiamo un metodo

```
void setSeparatoreTo(char sep)
```

che modifichi il carattere utilizzato dal metodo `toString` per separare ore e minuti.

(2) Possibilità di indicare gli orari su 12 ore invece che su 24 (`5:22pm`)

Aggiungiamo un metodo

```
void setFormato24(boolean b)
```

che consenta di selezionare il formato su 24 ore (se `b` è `true`) o su 12 ore (se `b` è `false`).

Miglioramenti

(1) Possibilità di selezionare un separatore diverso da `':'` tra ore e minuti

Aggiungiamo un metodo

```
void setSeparatoreTo(char sep)
```

che modifichi il carattere utilizzato dal metodo `toString` per separare ore e minuti.

(2) Possibilità di indicare gli orari su 12 ore invece che su 24 (`5:22pm`)

Aggiungiamo un metodo

```
void setFormato24(boolean b)
```

che consenta di selezionare il formato su 24 ore (se `b` è `true`) o su 12 ore (se `b` è `false`).

Questi metodi devono modificare una proprietà (il modo in cui vengono visualizzati gli oggetti) comune a tutte le istanze e quindi alla classe.

Orario: campi e metodi statici

- È necessario memorizzare il carattere utilizzato come separatore

Orario: campi e metodi statici

- È necessario memorizzare il carattere utilizzato come separatore
- Dato che è un'informazione comune all'intera classe lo memorizziamo in un campo statico.

```
public class Orario {
  //CAMPI STATICI
  private static char separatore = ':';

  //CAMPI
  private int ore, min;

  ...
```

Orario: miglioramenti

```
public String toString() {
  String stringaMinuti = (min < 10 ? "0" : "") + min;
  return String.valueOf(ore) + separatore + stringaMinuti;
}
```

Orario: miglioramenti

```
public String toString() {
  String stringaMinuti = (min < 10 ? "0" : "") + min;
  return String.valueOf(ore) + separatore + stringaMinuti;
}

public static void setSeparatoreTo(char ch) {
  separatore = ch;
}
```

Orario: miglioramenti

```
public String toString() {
  String stringaMinuti = (min < 10 ? "0" : "") + min;
  return String.valueOf(ore) + separatore + stringaMinuti;
}

public static void setSeparatoreTo(char ch) {
  separatore = ch;
}

public static char getSeparatore() {
  return separatore;
}
```

Orario: miglioramenti

```
//CAMPI STATICI
private static char separatore = ':';
private static boolean formato24 = true;

...
```

Orario: miglioramenti

```
//CAMPI STATICI
private static char separatore = ':';
private static boolean formato24 = true;

...

public static void setFormato24(boolean b) {
  formato24 = b;
}
```

Orario: miglioramenti

```
//CAMPI STATICI
private static char separatore = ':';
private static boolean formato24 = true;

...

public static void setFormato24(boolean b) {
  formato24 = b;
}

public static boolean isFormato24Attivo() {
  return formato24;
}
```

Orario: toString

```
public String toString() {
  String risultato;
  String stringaMin = (min < 10 ? "0" : "") + min;
  if (formato24)
    risultato = String.valueOf(ore) + separatore
                + stringaMin;
  else {
    ...
  }
  return risultato;
}
```

Orario: toString

```
else {
  int oraRisultato;
  String suff;
  if (ore == 0) {
    oraRisultato = 12;
    suff = "am";
  } else if (ore > 0 && ore < 12) {
    oraRisultato = ore;
    suff = "am";
  } else if (ore == 12) {
    oraRisultato = 12;
    suff = "pm";
  } else {
    oraRisultato = ore - 12;
    suff = "pm";
  }
  risultato = String.valueOf(oraRisultato) + separatore
              + stringaMin + suff;
}
```

Package

Permettono di raggruppare in unità logiche classi e interfacce correlate tra loro.

Package

Permettono di raggruppare in unità logiche classi e interfacce correlate tra loro.

Unità di compilazione

I file che contengono codice sorgente Java.

Package

Permettono di raggruppare in unità logiche classi e interfacce correlate tra loro.

Unità di compilazione

I file che contengono codice sorgente Java.

- Un'unità di compilazione può contenere diverse classi e interfacce

Package

Permettono di raggruppare in unità logiche classi e interfacce correlate tra loro.

Unità di compilazione

I file che contengono codice sorgente Java.

- Un'unità di compilazione può contenere diverse classi e interfacce
- Ogni unità può contenere al più una classe o un'interfaccia dichiarata `public`

Package

Permettono di raggruppare in unità logiche classi e interfacce correlate tra loro.

Unità di compilazione

I file che contengono codice sorgente Java.

- Un'unità di compilazione può contenere diverse classi e interfacce
- Ogni unità può contenere al più una classe o un'interfaccia dichiarata `public`
- Se una classe o interfaccia è dichiarata `public` essa determina il nome del file

Definizione di package

Un'unità di compilazione è aggiunta ad un package mediante l'istruzione

`package` *nome_package*

Deve obbligatoriamente essere la prima istruzione nell'unità di compilazione

Definizione di package

Un'unità di compilazione è aggiunta ad un package mediante l'istruzione

`package` *nome_package*

Deve obbligatoriamente essere la prima istruzione nell'unità di compilazione

Esempio

Per aggiungere le classi `Orario` e `Frazione` al package `prog.utili` fornito con il testo dobbiamo inserire l'istruzione

```
package prog.utili;
```

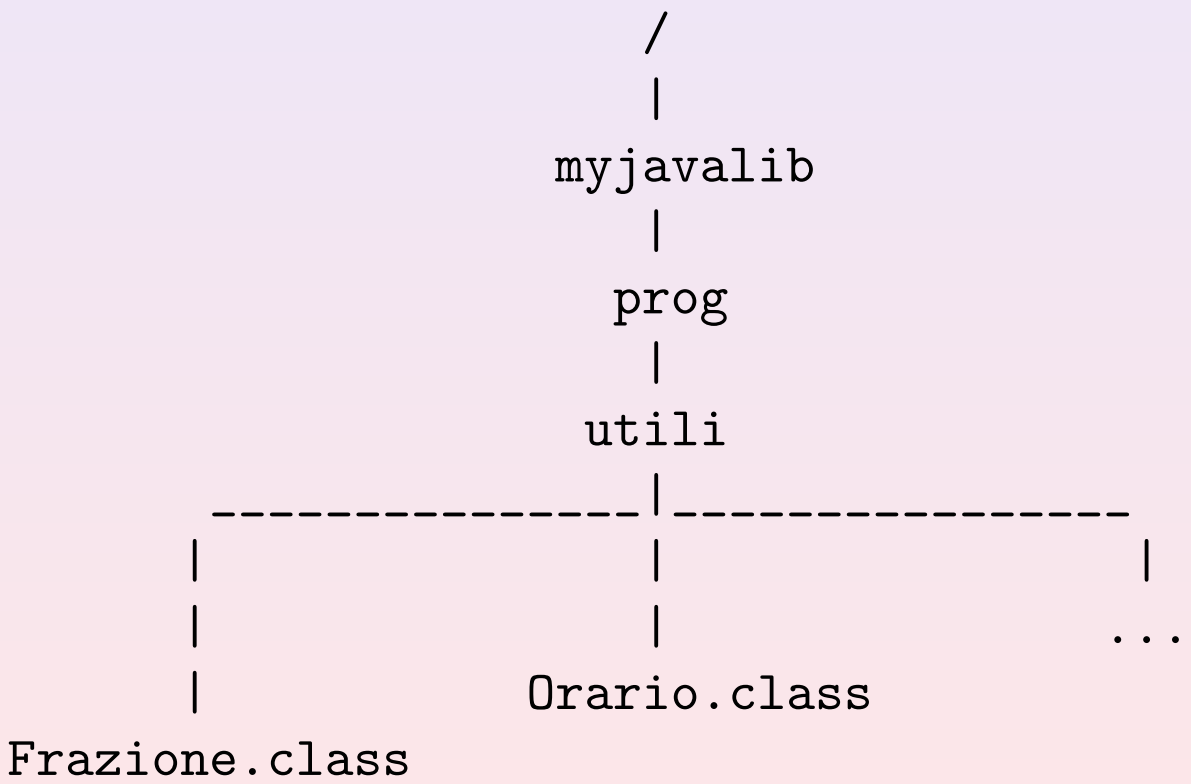
all'inizio dei file `Frazione.java` e `Orario.java`

Package e struttura delle directory

```
CLASSPATH = /myjavalib
```

Package e struttura delle directory

```
CLASSPATH = /myjavalib
```



Esempio

```
package pack1;

public class A {

  public String toString(){
    return "classe pack1.A";
  }
}
```

Esempio

```
package pack1;

public class A {

  public String toString(){
    return "classe pack1.A";
  }
}
```

Compilazione

```
> javac A.java
```

- viene generato il file `A.class`

Esempio

```
package pack1;

public class A {

  public String toString(){
    return "classe pack1.A";
  }
}
```

Compilazione

```
> javac A.java
```

- viene generato il file `A.class`
- il nome completo della classe è `pack1.A`

Utilizzo del nome completo

Si può utilizzare una classe indicandone il nome completo e omettendo la direttiva di importazione:

```
class Prova {
  public static void main(String[] args) {
    prog.io.ConsoleOutputManager out =
      new prog.io.ConsoleOutputManager();

    pack1.A a = new pack1.A();
    out.println(a.toString());
  }
}
```

Utilizzo del nome completo

Si può utilizzare una classe indicandone il nome completo e omettendo la direttiva di importazione:

```
class Prova {
  public static void main(String[] args) {
    prog.io.ConsoleOutputManager out =
      new prog.io.ConsoleOutputManager();

    pack1.A a = new pack1.A();
    out.println(a.toString());
  }
}
```

Le direttive di importazione aggiungono lo spazio dei nomi di un package, cioè i nomi delle classi e delle interfacce del package, a quanto il compilatore prende in considerazione

Conflitto sui nomi

Utilizzando i nomi completi è possibile utilizzare nel codice classi diverse con lo stesso nome.

```
import prog.io.*;

class Prova {

  public static void main(String[] args) {
    ConsoleOutputManager out = new ConsoleOutputManager();

    pack1.A a1 = new pack1.A();
    out.println(a1.toString());

    pack2.A a2 = new pack2.A();
    out.println(a2.toString());
  }
}
```

Importazione su richiesta

- Direttive di importazione che utilizzano il carattere *

Importazione su richiesta

- Direttive di importazione che utilizzano il carattere *

Esempio

```
import prog.io.*
```

Importazione su richiesta

- Direttive di importazione che utilizzano il carattere *

Esempio

```
import prog.io.*
```

- Aggiungono allo spazio dei nomi, i nomi di tutte le classi e di tutte le interfacce presenti nel package

Package di default

- Il compilatore importa automaticamente le classi e le interfacce del package `java.lang` e del package di default

Package di default

- Il compilatore importa automaticamente le classi e le interfacce del package `java.lang` e del package di default
- Package di default
  Un package definito automaticamente, senza nome, che include tutte le classi e tutte le interfacce definite nella directory di compilazione che non sono esplicitamente incluse in un package

Modificatori di visibilità

I package forniscono un ambiente di protezione in cui sviluppare codice nascondendone i dettagli all'esterno.

Modificatori di visibilità

I package forniscono un ambiente di protezione in cui sviluppare codice nascondendone i dettagli all'esterno.

Tale meccanismo di protezione è realizzato tramite i modificatori di visibilità.

Modificatori di visibilità

I package forniscono un ambiente di protezione in cui sviluppare codice nascondendone i dettagli all'esterno.

Tale meccanismo di protezione è realizzato tramite i modificatori di visibilità.

- `private` (membri)
  Ne limita la visibilità all'interno del codice della classe stessa

Modificatori di visibilità

I package forniscono un ambiente di protezione in cui sviluppare codice nascondendone i dettagli all'esterno.

Tale meccanismo di protezione è realizzato tramite i modificatori di visibilità.

- `private` (membri)
  Ne limita la visibilità all'interno del codice della classe stessa
- `public` (membri, classi, interfacce)
  Visibili ovunque

Modificatori di visibilità

I package forniscono un ambiente di protezione in cui sviluppare codice nascondendone i dettagli all'esterno.

Tale meccanismo di protezione è realizzato tramite i modificatori di visibilità.

- `private` (membri)
  Ne limita la visibilità all'interno del codice della classe stessa
- `public` (membri, classi, interfacce)
  Visibili ovunque
- *amichevole* (membri, classi, interfacce)
  Visibile all'interno del package

Modificatori di visibilità

I package forniscono un ambiente di protezione in cui sviluppare codice nascondendone i dettagli all'esterno.

Tale meccanismo di protezione è realizzato tramite i modificatori di visibilità.

- `private` (membri)
  Ne limita la visibilità all'interno del codice della classe stessa
- `public` (membri, classi, interfacce)
  Visibili ovunque
- *amichevole* (membri, classi, interfacce)
  Visibile all'interno del package
- `protected`
  Il suo significato è legato all'estensione delle classi