

Realizzare classi

ESEMPIO

```
Contatore c = new Contatore ();  
c.click();  
c.click();  
int result = c.getValue();  
//assegna a result il valore 2.
```

Ogni contatore deve memorizzare una variabile che tenga traccia del numero di clic simulati sui pulsanti.



Definizioni:

Le variabili di esemplare (o di istanza) memorizzano i dati di un oggetto.

Un esemplare (istanza) di una classe: un oggetto della classe.

Una variabile di istanza è una posizione di archiviazione presente in ogni oggetto della classe.

La dichiarazione di classe specifica le variabili di esemplare o istanza:

```
public class Contatore
{
    private int value;
    ...
}
```

Le variabili di istanza di un oggetto memorizzano i dati richiesti per l'esecuzione dei suoi metodi.

Dichiarazione di variabili di esemplare

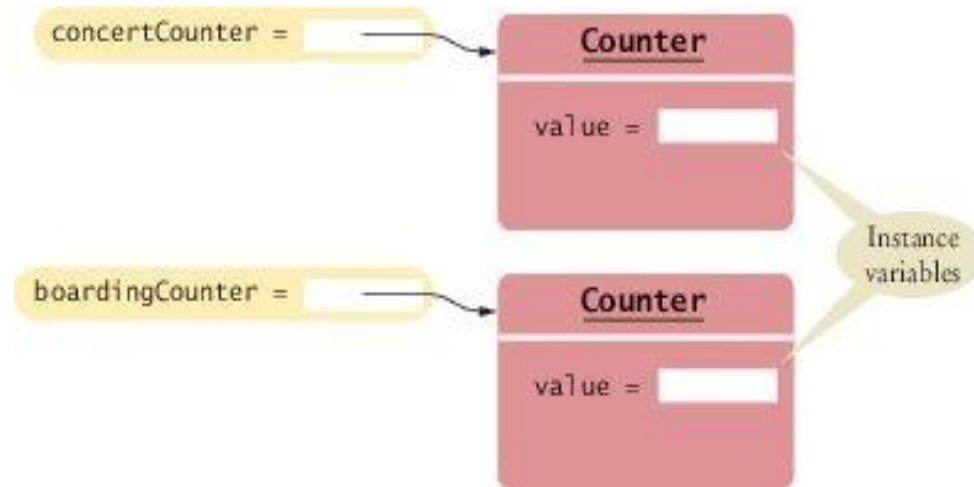
Una dichiarazione di istanza è composta dalle seguenti parti:

- Modalità di accesso (private)
- tipo di variabile (come int)
- nome della variabile (come value)

Dovresti dichiarare tutte le variabili di esemplare (istanza) come private.

Dichiarazione di variabili di esemplare

Ogni oggetto di una classe ha il proprio insieme di variabili di esemplare.



Dichiarazione di variabili di esemplare

Syntax

```
public class ClassName
{
    private typeName variableName;
    . . .
}
```

Instance variables should
always be private.

```
public class Counter
{
    private int value;
    . . .
}
```

Each object of this class
has a separate copy of
this instance variable.

Type of the variable

Variabili di esemplare

Questi orologi hanno un comportamento comune, ma ognuno di essi ha uno stato diverso.

Allo stesso modo, gli oggetti di una classe possono avere le variabili di esemplare impostate su valori diversi.



Metodi

il metodo del clic fa aumentare il valore del contatore di 1:

```
public void click()  
{  
    value = value + 1;  
}
```

Influisce sul valore della variabile di esemplare dell'oggetto su cui viene invocato il metodo

Il metodo chiama `concertCounter.click();`

Aumenta la variabile valore dell'oggetto `concertCounter`

Metodi

Il metodo `getValue` restituisce il valore corrente:

```
public int getValue()  
{  
    return value;  
}
```

La dichiarazione del `return`:

- Termina la chiamata al metodo
- Restituisce un risultato (il valore restituito) al chiamante del metodo

È possibile accedere alle variabili di esemplare private solo con metodi della stessa classe.



Incapsulamento:

L'incapsulamento è il processo per nascondere i dettagli di implementazione e fornire metodi per l'accesso ai dati.

Per incapsulare i dati:

- Dichiarare le variabili di esemplare come private
- Dichiarare i metodi pubblici che accedono alle variabili

L'incapsulamento consente a un programmatore di utilizzare una classe senza doverne conoscere l'implementazione.

L'occultamento delle informazioni rende più semplice per l'implementatore di una classe individuare gli errori e modificare le implementazioni.

Incapsulamento:



Un termostato funziona come una "scatola nera" i cui meccanismi interni sono nascosti.

Quando usiamo classi, come Rectangle e String, in programmi, siamo come un appaltatore che installa un termostato.

Quando implementiamo le nostre classi siamo come il produttore che mette insieme un termostato senza parti.

```
1  /**
2  Questa classe modella un contatore.
3  */
4  public class Counter
5  {
6      private int value;
7
8  /**
9  Restituisce il valore corrente di
   value
10
11 */
12     public int getValue()
13     {
14         return value;
15     }
16
```

```
17 /**
18 Aumenta il contatore di 1.
19 */
20 public void click()
21 {
22     value = value + 1;
23 }
24
25 /**
26 Resetta a 0 il contatore.
27 */
28 public void reset()
29 {
30     value = 0;
31 }
32 }
```

Esercizio:

Fornisci il corpo di un metodo

```
public void unclick()
```

che annulla il clic indesiderato di un pulsante.

Esercizio:

Supponiamo di utilizzare una classe Clock con variabili di istanza private ore e minuti. Come puoi accedere a queste variabili nel tuo programma?

Esercizio:

Supponiamo di lavorare in un'azienda che produce software di finanza personale. Ti viene chiesto di progettare e implementare una classe per la rappresentazione di conti bancari. Chi saranno gli utenti della tua classe?

Progettare un'interfaccia pubblica di una classe:

Per implementare una classe, devi prima sapere quali metodi sono richiesti.

Comportamento essenziale di un conto bancario:

- depositare denaro
- prelevare denaro
- ottenere il saldo attuale

Progettare un'interfaccia pubblica di una classe:

```
harrysChecking.deposit(2000);  
harrysChecking.withdraw(500);  
System.out.println(harrysChecking.getBalance());
```

Ecco le intestazioni dei metodi necessarie per una classe BankAccount:

```
public void deposit(double amount)  
public void withdraw(double amount)  
public double getBalance()
```

Progettare un'interfaccia pubblica di una classe:

Il corpo di un metodo: costituito da istruzioni che vengono eseguite quando viene chiamato il metodo

```
public void deposit(double amount)
{
    implementazione
}
```

Puoi compilare il corpo del metodo in modo che venga compilato

```
public double getBalance()
{
    //Implementazione
    return 0.0;
}
```

Metodi

I metodi di BankAccount sono stati dichiarati pubblici.

I metodi pubblici possono essere chiamati da tutti gli altri metodi del programma.

I metodi possono anche essere dichiarati privati.

I metodi privati possono essere chiamati solo da altri metodi nella stessa classe

I metodi privati non fanno parte dell'interfaccia pubblica

Definire un costruttore

In Java, un costruttore è molto simile a un metodo ma ha due differenze importanti:

- 1) Ha lo stesso nome della classe
- 2) I costruttori non definiscono un tipo per il valore restituito

→ Inizializza oggetti

→ Imposta i dati iniziali per gli oggetti

Definire un costruttore:

Due costruttori:

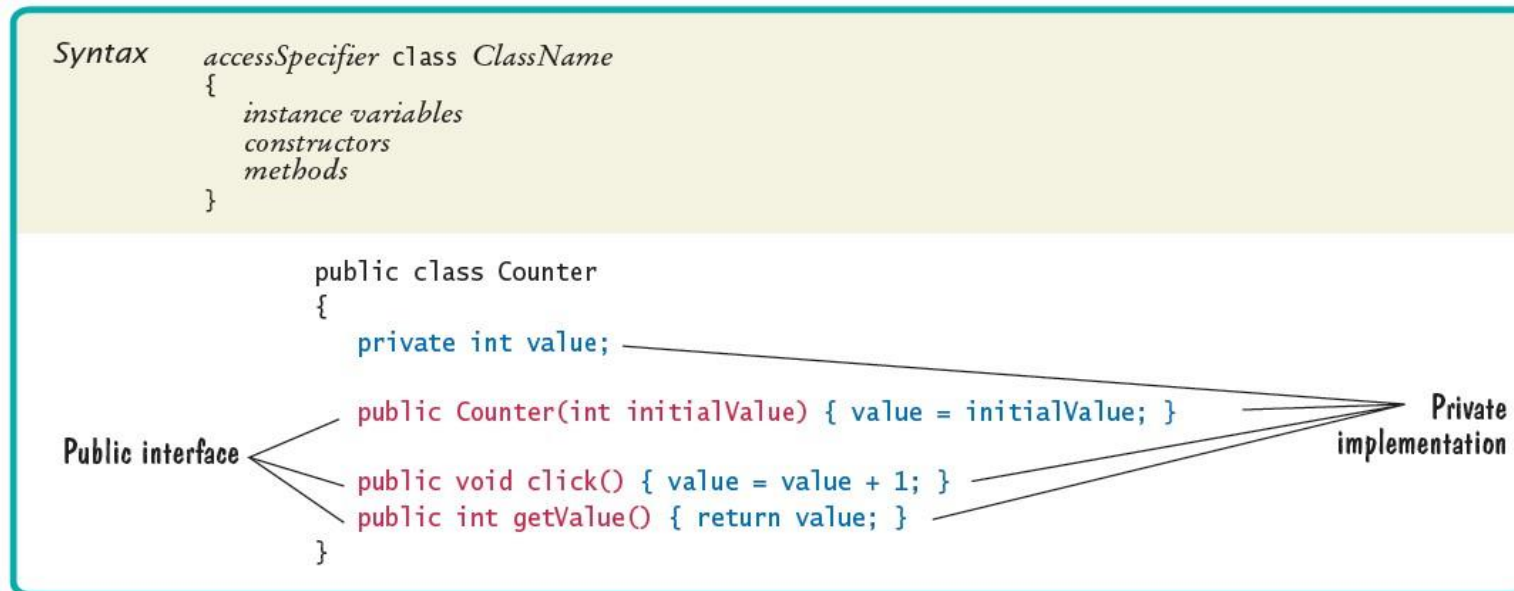
- `public BankAccount()`
- `public BankAccount(double initialBalance)`

Uso:

- `BankAccount harrysChecking = new BankAccount();`
- `BankAccount momsSavings = new BankAccount(5000);`

Dichiarare una classe:

i costruttori pubblici e i metodi di una classe formano l'interfaccia pubblica della classe. Queste sono le operazioni che ogni programmatore può usare



Commentare l'interfaccia pubblica di una classe:

Usa i commenti della documentazione per descrivere le classi e i metodi pubblici dei tuoi programmi.

Java ha un modulo standard per i commenti sulla documentazione.

Un programma chiamato javadoc può generare automaticamente un insieme di pagine HTML.

Il commento di documentazione deve essere posto prima della dichiarazione di classe o metodo che viene documentata

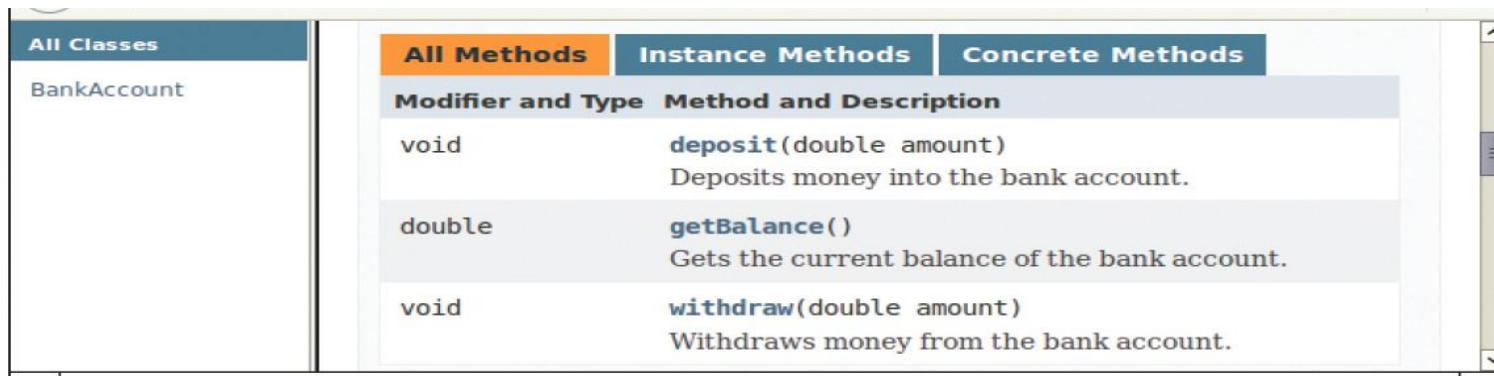
Commentare l'interfaccia pubblica di una classe:

```
/**
 * Preleva un importo dal conto bancario
 * @param amount: importo da prelevare
 */
public void withdraw(double amount)
{
    implementazione
}

/**
 * Restituisce il saldo del conto corrente
 * @return il saldo attuale
 */
public double getBalance()
{
    implementazione
}
```


Commentare l'interfaccia pubblica di una classe:

```
/**  
Un conto bancario ha un saldo che può cambiare tramite  
versamenti e prelievi.  
*/  
public class BankAccount  
{  
    . . .  
}
```



The screenshot shows an IDE window with a tab titled 'All Classes' containing 'BankAccount'. The 'All Methods' tab is selected, displaying a table of methods.

Modifier and Type	Method and Description
void	deposit (double amount) Deposits money into the bank account.
double	getBalance () Gets the current balance of the bank account.
void	withdraw (double amount) Withdraws money from the bank account.

Esercizio

Come puoi usare i metodi dell'interfaccia pubblica per svuotare il `harrysChecking?` conto bancario?

Esercizio

Cosa c'è di sbagliato?

```
BankAccount harrysChecking = new BankAccount(10000);  
System.out.println(harrysChecking.withdraw(500));
```

Esercizio

Supponiamo di volere qualcosa in più dal conto bancario qualcosa che tenga traccia di un numero di conto oltre al saldo. Come cambieresti l'interfaccia pubblica per accogliere questo miglioramento?

Esercizio

Supponiamo di migliorare la classe BankAccount in modo che ogni conto abbia un numero di conto. Fornire un commento alla documentazione per il costruttore

```
public BankAccount (int accountNumber, double initialBalance)
```

Esercizio

Cosa non va?

```
/**  
Ogni conto ha un numero di conto.  
@return il numero di conto del conto  
*/  
public int getAccountNumber()
```

Realizzare una classe:

L'implementazione privata di una classe consiste in:

- variabili di istanza
- i corpi dei costruttori
- i corpi dei metodi

Realizzare una classe: definire le variabili di istanza

Determinare i dati contenuti in ciascun oggetto conto bancario.

Cosa deve ricordare l'oggetto per poter eseguire i suoi metodi?

Ogni oggetto del conto bancario deve solo memorizzare il saldo corrente (nel nostro esempio).

Dichiarazione della variabile di istanza BankAccount:

```
public class BankAccount
{
    private double balance;
    // Methods and constructors
    below
    . . .
}
```


Realizzare una classe: definire le variabili di istanza

Come un esploratore ha bisogno di trasportare tutti gli elementi che potrebbero essere necessari per la sua avventura, un oggetto deve memorizzare i dati richiesti per le sue chiamate al metodo.

Quindi dovete soffermarvi su capire cosa serve all'oggetto

Realizzare una classe: definire i costruttori

Il compito del costruttore è inizializzare le variabili di istanza dell'oggetto.

Il costruttore senza argomenti imposta il saldo su zero.

```
public BankAccount()  
{  
    balance = 0;  
}
```

Il secondo costruttore imposta il saldo sul valore fornito come argomento di costruzione.

```
public BankAccount(double initialBalance)  
{  
    balance = initialBalance;  
}
```

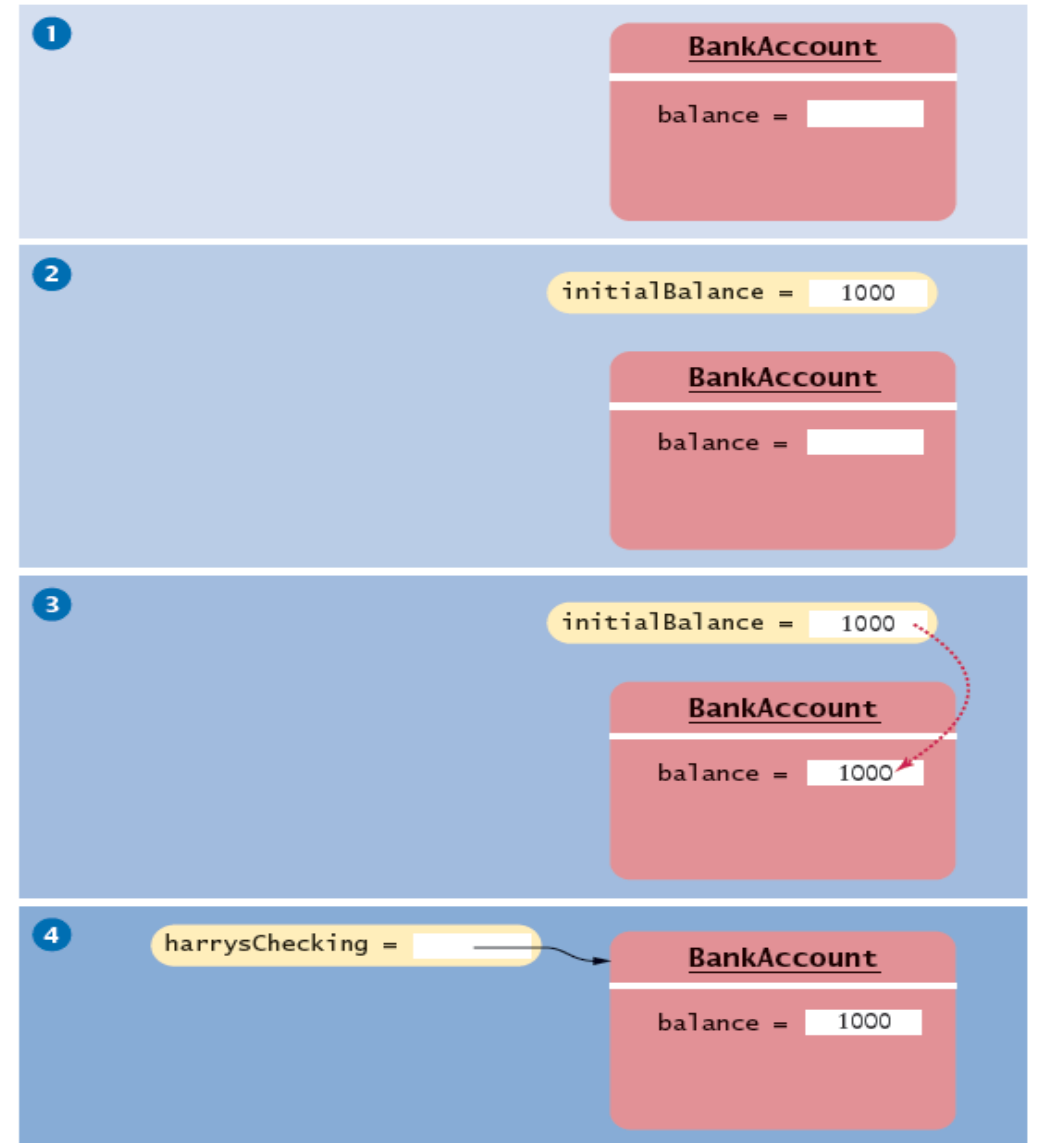
Realizzare una classe: definire i costruttori

Passaggi eseguiti quando viene eseguita la seguente istruzione:

```
BankAccount harrysChecking = new BankAccount(1000);
```

1. Crea un nuovo oggetto di tipo BankAccount.
2. Richiama il secondo costruttore perché viene fornito un argomento nella chiamata del costruttore
3. Imposta il parametro variabile initialBalance uguale a 1000.
4. Imposta la variabile di istanza balance dell'oggetto appena creato su initialBalance.
5. Restituisce un riferimento all'oggetto, ovvero la posizione di memoria dell'oggetto.
6. Memorizza quel riferimento all'oggetto nella variabile harrysChecking.

Realizzare una classe: definire i costruttori



Realizzare una classe: definire i metodi

Il metodo è di accesso o di modifica?

Metodo di modifica:

Aggiorna le variabili di istanza in qualche modo

Metodo di accesso:

Recupera o calcola un risultato

metodo deposit? - un metodo di modifica

Aggiorna il saldo

```
public void deposit(double amount)
{
    balance = balance + amount;
}
```

Realizzare una classe: definire i metodi

metodo withdraw? - un metodo di modifica

```
public void withdraw(double amount)
{
    balance = balance - amount;
}
```

metodo getBalance? - un metodo di accesso

```
public double getBalance()
{
    return balance;
}
```

Realizzare una classe: definire i metodi

Table 1 Implementing Classes

Example	Comments
<code>public class BankAccount { . . . }</code>	This is the start of a class declaration. Instance variables, methods, and constructors are placed inside the braces.
<code>private double balance;</code>	This is an instance variable of type double. Instance variables should be declared as private.
<code>public double getBalance() { . . . }</code>	This is a method declaration. The body of the method must be placed inside the braces.
<code>. . . { return balance; }</code>	This is the body of the getBalance method. The return statement returns a value to the caller of the method.
<code>public void deposit(double amount) { . . . }</code>	This is a method with a parameter variable (amount). Because the method is declared as void, it has no return value.
<code>. . . { balance = balance + amount; }</code>	This is the body of the deposit method. It does not have a return statement.
<code>public BankAccount() { . . . }</code>	This is a constructor declaration. A constructor has the same name as the class and no return type.
<code>. . . { balance = 0; }</code>	This is the body of the constructor. A constructor should initialize the instance variables.




Esercizio

- Supponiamo di modificare la classe BankAccount in modo che ogni conto bancario abbia un numero di conto. In che modo questa modifica influisce sulle variabili di istanza?

Esercizio


Perché il seguente codice non riesce a derubare il conto bancario della mamma?

```
public class BankRobber
{
    public static void main(String[] args)
    {
        BankAccount momsSavings = new
        BankAccount(1000);
        momsSavings.balance = 0;
    }
}
```



Unit test

- BankAccount.java non può essere eseguito: Non ha un metodo principale
- La maggior parte delle classi non ha un metodo principale
- Prima di utilizzare BankAccount.java in un programma più grande: Dovresti fare un test isolato
- Unit test: verifica che una classe funzioni correttamente isolatamente, al di fuori di un programma completo.



Unit test

Per testare una classe, si utilizza un ambiente per i test interattivi o si scrive una classe di tester per eseguire le istruzioni di test.

Classe tester: una classe con un metodo principale che contiene istruzioni per testare un'altra classe. In genere esegue i seguenti passaggi:

- Costruisce uno o più oggetti della classe che viene testata

- Invoca uno o più metodi

- Stampa uno o più risultati

- Stampa i risultati attesi

Unit test

```
1 /**
2  A class to test the BankAccount class.
3  */
4 public class BankAccountTester
5 {
6     /**
7     Tests the methods of the BankAccount class.
8     @param args not used
9     */
10    public static void main(String[] args)
11    {
12        BankAccount harrysChecking = new BankAccount();
13        harrysChecking.deposit(2000);
14        harrysChecking.withdraw(500);
15        System.out.println(harrysChecking.getBalance());
16        System.out.println("Expected: 1500");
17    }
18 }
```

Unit test-costruire un programma

- Per produrre un programma: combinare sia BankAccount che BankAccountTester.
- I dettagli per la creazione del programma variano. Nella maggior parte degli ambienti, è necessario eseguire questi passaggi:
- Crea una nuova sottocartella per il tuo programma Crea due file, uno per ogni classe
- Compila entrambi i file
- Esegui il programma di prova
- BankAccount e BankAccountTest hanno scopi completamente diversi:
La classe BankAccount descrive gli oggetti che calcolano i saldi bancari
La classe BankAccountTester esegue test che mettono alla prova un oggetto BankAccount

Esercizio

Quando esegui il programma BankAccountTester, quanti oggetti di classe BankAccount sono costruiti? Quanti oggetti di tipo BankAccountTester?

Esercitazione

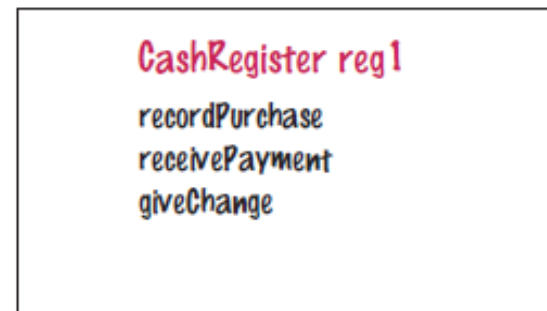
Classe CashRegister

Problem solving

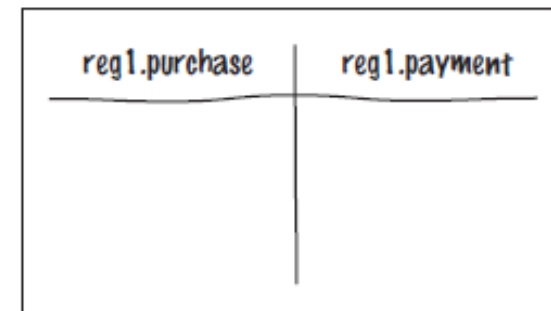
- Abilità importante: la capacità di simulare le azioni di un programma con carta e matita.
 - Usa una scheda o una foglio adesivo per ogni oggetto:
 - Scrivi i metodi sul davanti
 - Crea una tabella per i valori delle variabili di istanza sul retro
- Classe CashRegister

Problem solving

Classe CashRegister



front



back

Problem solving

Quando viene costruito un oggetto, inserire i valori iniziali delle variabili di istanza.

Aggiorna i valori delle variabili di istanza quando viene chiamato un metodo di modifica. Dopo una chiamata al metodo recordPurchase di cashRegister

Più di un oggetto: crea più fogli

reg1.purchase	reg1.payment
0	0

reg1.purchase	reg1.payment
0 19.95	0

reg1.purchase	reg1.payment
0 19.95	0 19.95

reg2.purchase	reg2.payment
0 29.50 9.25	0 50.00

Problem solving

Utile quando si migliora una classe..

Migliora la classe CashRegister per calcolare l'imposta sulle vendite. Aggiungi i metodi recordTaxablePurchase e getSalesTax nella parte anteriore della carta.

Non disponi di informazioni sufficienti per calcolare l'imposta sulle vendite:

- bisogno di aliquota fiscale (tax)

- bisogno totale degli elementi imponibili (taxable)

Hai bisogno di variabili di istanza aggiuntive per:

- taxRate

- taxablePurchase

Problem solving

Esempio:

```
CashRegister reg3(7.5); // 7.5 percent sales tax  
reg3.recordPurchase(3.95); // Not taxable  
reg3.recordTaxablePurchase(19.95); // Taxable
```

reg3.purchase	reg3.taxablePurchase	reg3.payment	reg3.taxRate
0 3.95	0 19.95	0	7.5

Variabili locali

Le variabili locali sono dichiarate nel corpo di un metodo:

```
public double giveChange()  
{  
    double change = payment - purchase;  
    purchase = 0;  
    payment = 0;  
    return change;  
}
```

Quando un metodo «esce», le sue variabili locali vengono rimosse.

Le variabili dei parametri sono dichiarate nell'intestazione di un metodo:

```
public void enterPayment(double amount)
```

Variabili locali

Le variabili locali e parametriche appartengono ai metodi:

- Quando un metodo viene eseguito, le sue variabili locali e dei parametri prendono vita
- Quando il metodo esce, vengono rimosse immediatamente

Le variabili di istanza appartengono agli oggetti, non ai metodi:

- Quando viene costruito un oggetto, vengono create le sue variabili di istanza
- Le variabili di istanza rimangono attive finché nessun metodo utilizza più l'oggetto

Le variabili di istanza sono inizializzate su un valore predefinito:

- I numeri sono inizializzati a 0
- I riferimenti agli oggetti sono impostati su un valore speciale chiamato null (Un riferimento a un oggetto null si riferisce a nessun oggetto)

NB: Devi inizializzare le variabili locali: Il compilatore si lamenta se non lo fai

Esercizio

Cosa hanno in comune le variabili locali e le variabili dei parametri? In quale aspetto essenziale differiscono?

Il riferimento this

Quando viene chiamato un metodo vengono passati due tipi di input:

- L'oggetto con cui il metodo viene invocato
- Gli argomenti del metodo

Nella chiamata `momsSavings.deposit(500)` il metodo deve sapere:

- L'oggetto del conto (`momsSavings`)
- L'importo depositato (500)

Il parametro implicito di un metodo è l'oggetto su cui viene invocato il metodo. Tutte le altre variabili dei parametri sono chiamate parametri espliciti

Il riferimento this

```
public void deposit(double amount)
{
    balance = balance + amount;
}
```

amount è il parametro esplicito

Il parametro implicito (momSavings) non viene visualizzato

balance significa momSavings.balance

Quando fai riferimento a una variabile di istanza all'interno di un metodo, significa la variabile di istanza del parametro implicito.

Il riferimento this

Il this riferimento denota il parametro implicito

```
balance = balance + amount;
```

Ha il seguente significato:

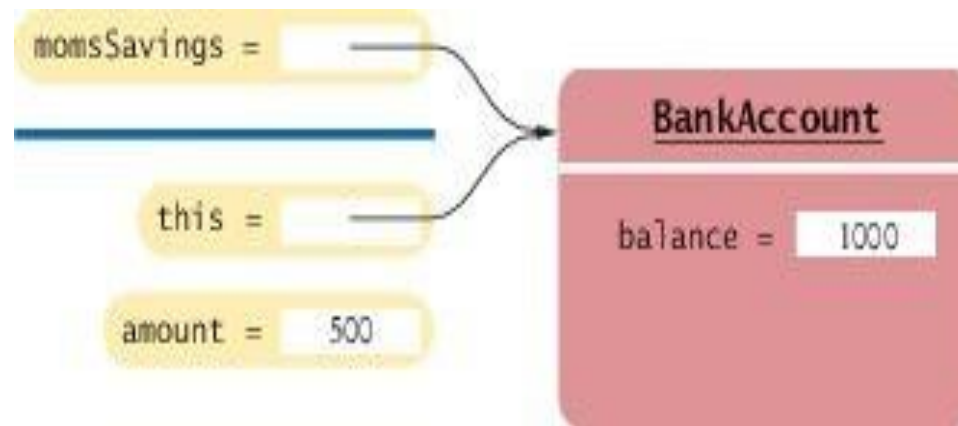
```
this.balance = this.balance + amount;
```

Quando fai riferimento a una variabile di istanza in un metodo, il compilatore la applica automaticamente a questo riferimento.

Il riferimento this

Alcuni programmatori ritengono che l'inserimento di `this` prima di ogni riferimento alla variabile di istanza renda il codice più chiaro:

```
public BankAccount(double initialBalance)
{
    this.balance = initialBalance;
}
```



Il riferimento this

this può essere utilizzato per distinguere tra variabili di istanza e variabili locali o di parametro:

```
public BankAccount(double balance)
{
    this.balance = balance;
}
```

Una variabile locale nasconde una variabile di istanza con lo stesso nome. È possibile accedere al nome della variabile di istanza tramite this. In Java, le variabili locali e dei parametri vengono considerate per prime quando si cercano i nomi delle variabili.

Quindi

```
this.balance = balance;
```

significa: "Imposta il saldo della variabile di istanza sul saldo della variabile del parametro".

Il riferimento this

Una chiamata al metodo senza un parametro implicito viene applicata allo stesso oggetto. Esempio:

```
public class BankAccount
{
    . . .
    public void monthlyFee()
    {
        withdraw(10); // Withdraw $10 from this account
    }
}
```

Il parametro implicito del metodo di `withdraw` è il parametro implicito (invisibile) del metodo `MonthFee`

Puoi usare questo riferimento per rendere il metodo più facile da leggere:

Il riferimento this

Puoi usare questo riferimento per rendere il metodo più facile da leggere:

```
public class BankAccount
{
    . . .
    public void monthlyFee()
    {
        this.withdraw(10); // Withdraw $10 from this account
    }
}
```

Questo indica un prelievo dallo stesso oggetto di tipo conto bancario con cui si sta eseguendo l'operazione montlyFee