Lezione 4

Definizione di Classi Istanziabili

Lezione 4 - Obiettivi

Al termine di questa lezione saremo in grado di

- Definire una classe istanziabile con metodi e costruttori.
- Distinguere tra variabili locali e variabili dell'istanza.
- Definire e usare metodi che restituiscono un valore.
- Distinguere metodi pubblici e metodi privati.
- Distinguere attributi pubblici e attributi privati.
- Descrivere come gli argomenti sono passati ai parametri nelle definizioni dei metodi.
- Utilizzare System.out per visualizzare informazioni diagnostiche sul funzionamento dei programmi.

Progettazione ad Oggetti

- Imparare a definire classi istanziabili è il primo passo per acquisire le capacità necessarie a costruire applicazioni in Java.
- Una classe è *istanziabile* se è possibile creare istanze della classe. Le classi MainWindow, InputBox, e OutputBox sono tutte classi istanziabili mentre la classe Math non è istanziabile.
- In questa lezione impareremo a definire classi istanziabili e i diversi tipi di metodi in esse inclusi.

CurrencyConverter

- Un esempio per coprire gli elementi basilari della definizione di classi istanziabili.
- Nel progettare una classe istanziabile, si comincia dalla specifica della classe, ossia dalla descrizione del comportamento desiderato per la classe e le sue istanze.
- Un oggetto CurrencyConverter deve eseguire conversioni tra valute internazionali e il dollaro statunitense.
- Qual'è il modo più naturale di interagire con un oggetto CurrencyConverter?

Metodi fromDollar e toDollar

- Almeno due metodi di conversione sembrano necessari: fromDollar e toDollar.

```
CurrencyConverter     yenConverter;
yenConverter       = new CurrencyConverter;
double amountInYen
          = yenConverter.fromDollar( 500);

double amountInUS
          = yenConverter.toDollar(15000);
```

Converte 500 dollari in yen.

Converte 15,000 yen in dollari.

Metodo setExchangeRate

- Dato che il tasso di cambio fluttua, è necessario un metodo per impostare il tasso di cambio.

```
CurrencyConverter     yenConverter;
yenConverter       = new CurrencyConverter;
yenConverter.setExchangeRate( 106.55);
```

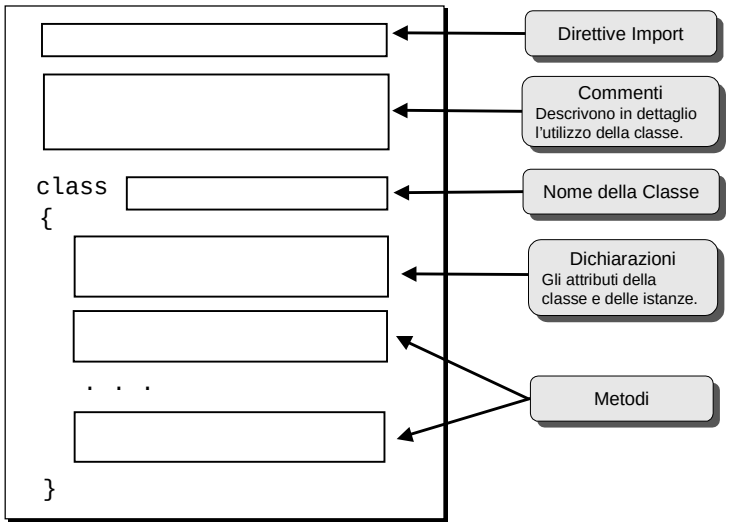
1.00 USD = 106.55 yen

Utilizzo di più Istanze

- Una volta definita la classe CurrencyConverter è possibile utilizzarne diverse istanze.

```
CurrencyConverter     yenConverter, euroConverter;
double                amountInYen, amountInEuro, amountInDollar;

yenConverter     = new CurrencyConverter();
yenConverter.setExchangeRate(130.77);

euroConverter = new CurrencyConverter( );
euroConverter.setExchangeRate(0.95);

amountInYen      = yenConverter.fromDollar( 200 );
amountInEuro     = markConverter.fromDollar( 200 );

amountInDollar   = yenConverter.toDollar( 10000 );
amountInEuro     = euroConverter.fromDollar(amountInDollar);
```


Schema per la Definizione di Classi

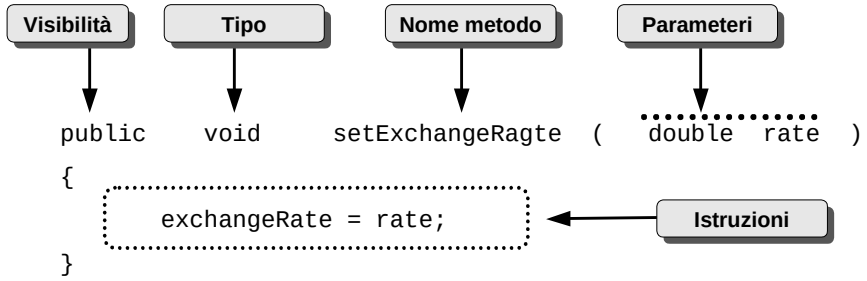


La Classe CurrencyConverter

```
/**
 * Questa classe è utilizzata per convertire importi tra
 * una valuta internazionale e il dollaro statunitense.
 *
 * @author Dr. Caffeine
 */
class CurrencyConverter
{
    /**
     * quanto vale 1.00 USD nella valuta internazionale
     */
    private double exchangeRate;

    //method declarations come here

}
```


Dichiarazione di Metodi

```
<modificatore>  <tipo>  <nome metodo>  ( <parametri>  )
{
       <istruzioni>
}
```



Metodi che Restituiscono un Valore

- I metodi che non restituiscono alcun valore hanno tipo void.
- I metodi che restituiscono un valore hanno tipo diverso da void.
- Un metodo che restituisce un valore deve includere un'istruzione nel seguente formato:

```
return  <espressione> ;
```

```
public double toDollar( double foreignMoney )
{
    return (foreignMoney / exchangeRate);
}
```


Commento di un Metodo in javadoc

```
/**
 * Converte una somma in dollari in una equivalente
 * somma in valuta internazionale.
 *
 * @param dollar l'ammontare di dollari da convertire
 *
 * @return ammontare in valuta internazionale
 */
```

Costruttori

- Un costruttore è un particolare metodo che viene eseguito ogni volta che una nuova istanza della classe viene creata.
- Lo scopo del costruttore è di inizializzare un oggetto in uno stato valido. Ogni volta che un oggetto viene creato, è necessario inizializzarne gli attributi nel costruttore.
- Il nome di un costruttore coincide con il nome della classe.
- Se nessun costruttore viene definito per una classe, allora il compilatore Java includerà un costruttore di default.

Costruttore di Default

- Il costruttore di default equivale al seguente:

```
public <class name> (   )
{

}
```

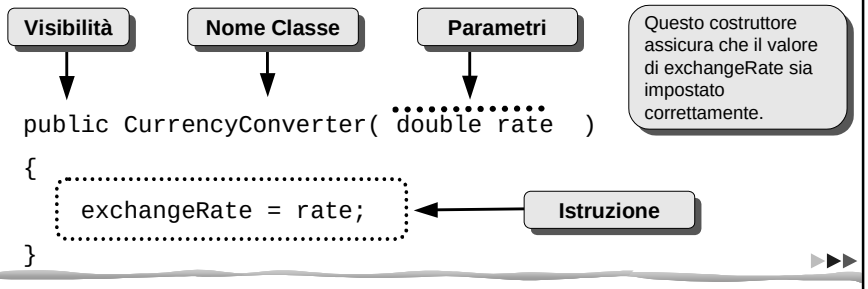
Un costruttore di default non ha alcuna istruzione nel suo corpo.

```
public CurrencyConverter(   )
{

}
```


Definizione di Costruttori

- Un costruttore ha il seguente formato:

```
public <nome classe> ( <parametri>  )
{
     <istruzioni>

}
```



Costruttori Multipli

- Una classe può includere più costruttori a patto che
  - Abbiano un diverso numero di parametri
  - Abbiano parametri di tipo diverso se il numero di parametri è lo stesso

```
public MyClass( int   value ) { … }
public MyClass(             ) { … }
public MyClass( float value ) { … }
```

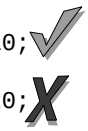
Questi costruttori non entrano in conflitto tra di loro.

Modificatori di Visibilità: public e private

- I modificatori public e private impostano l'accessibilità degli attributi e dei metodi
- Se il componente di una classe (metodo o attributo) viene dichiarato private, nessun metodo esterno può accedervi.
- Se il componente di una classe viene dichiarato public, qualsiasi metodo esterno può accedervi.

```
class Test
{
    public  int memberOne;
    private int memberTwo;
}
```

```
Test myTest = new MyTest();

myTest.memberOne = 10;

myTest.memberTwo = 20;
```



Gli Attributi Devono Essere private

- Dichiarare gli attributi (di classe e di istanza) private assicura l'integrità della classe.
- Gli attributi contengono i dettagli realizzativi di una classe e devono essere mantenuti invisibili all'esterno.
- Se un attributo è dichiarato public, non è possibile modificarne la realizzazione senza che questo comporti modifiche a tutte le classi che accedono direttamente all'attributo.
- Eccezione: Le costanti possono (dovrebbero) essere dichiarate public se è previsto il loro utilizzo da parte di metodi esterni.

Variabili Locali

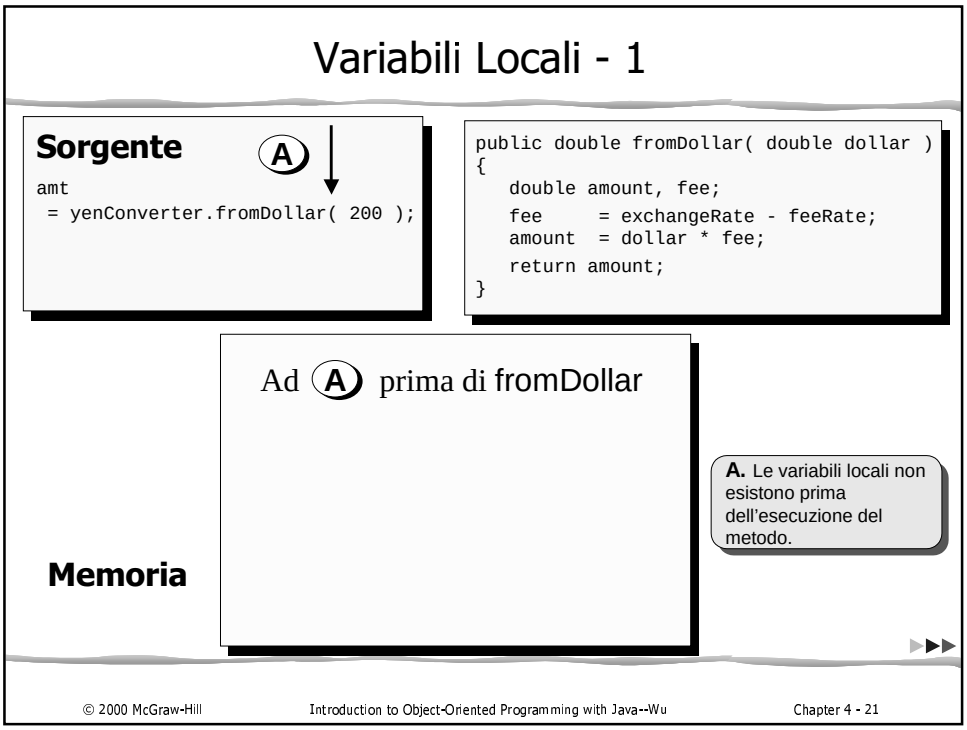
- Gli attributi sono condivisi tra tutti i metodi di una classe.
- Una variabile locale è una variabile dichiarata all'interno di una dichiarazione di un metodo.
- Le variabili locali sono accessibili solo all'interno del metodo in cui sono dichiarate.
- Lo spazio in memoria per le variabili locali è allocato solo durante l'esecuzione del metodo. Quando l'esecuzione del metodo è completa, lo spazio in memoria viene liberato.
- I parametri di un metodo sono locali al metodo.

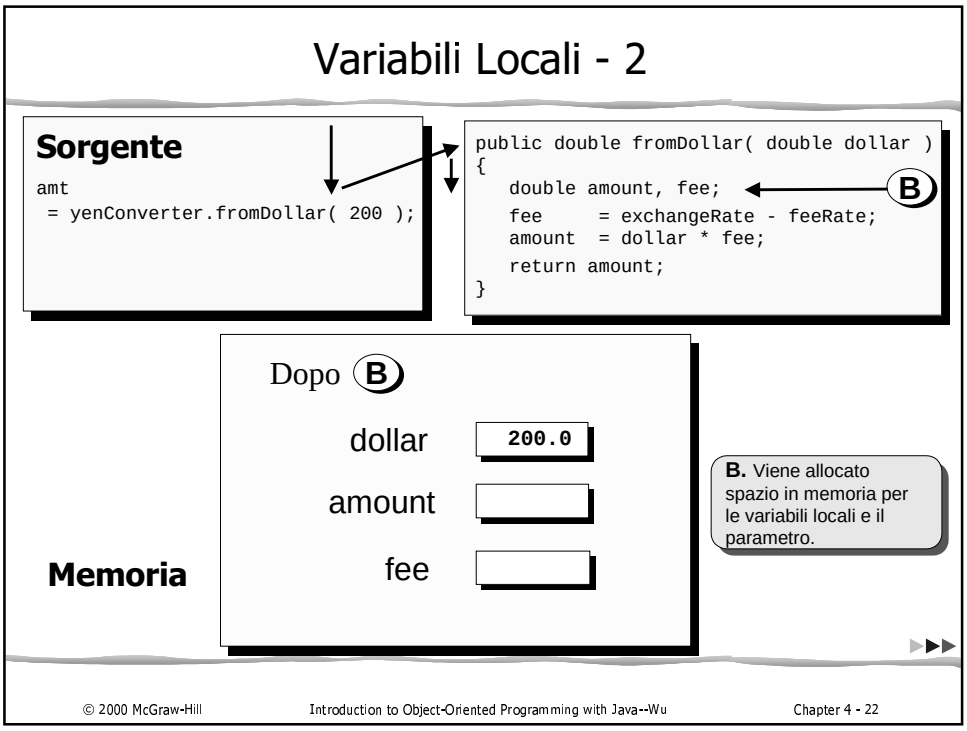
Esempio di Metodo con Variabili Locali

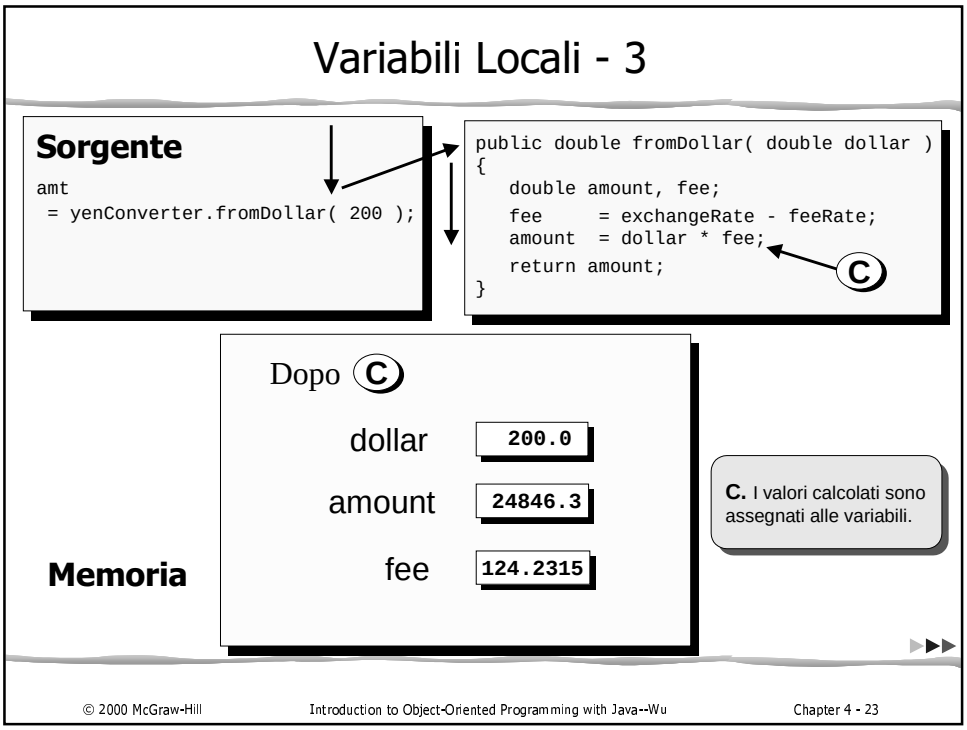
```
public double fromDollar( double   dollar )
{
        double   amount, fee;

        fee          = exchangeRate - feeRate;
        amount       = dollar * fee;

        return amount;
}
```

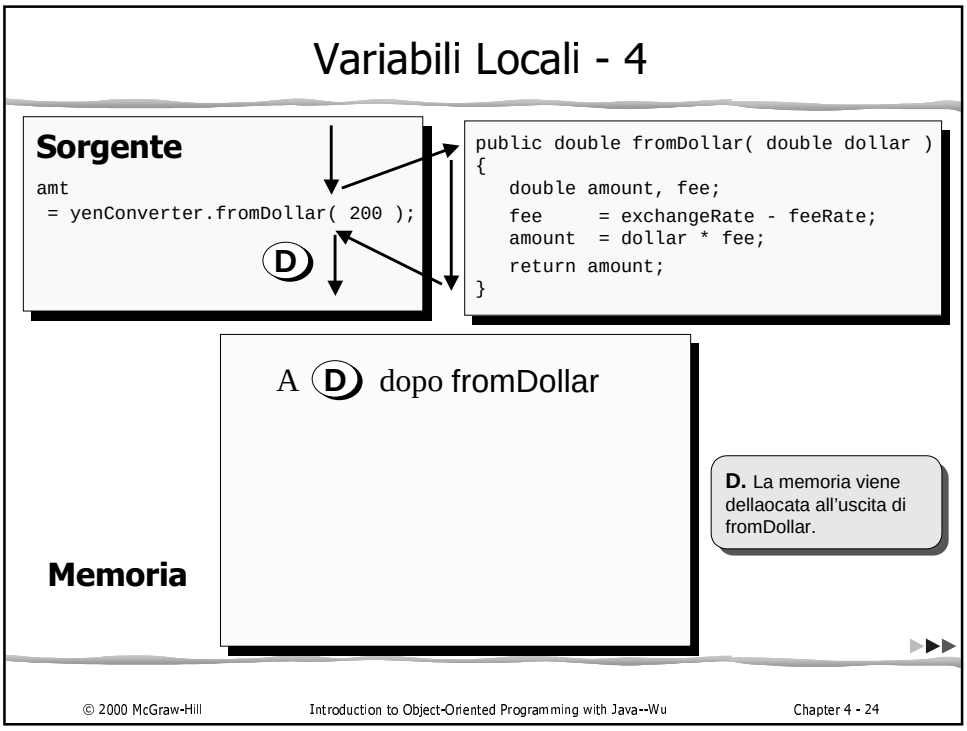
Parametro

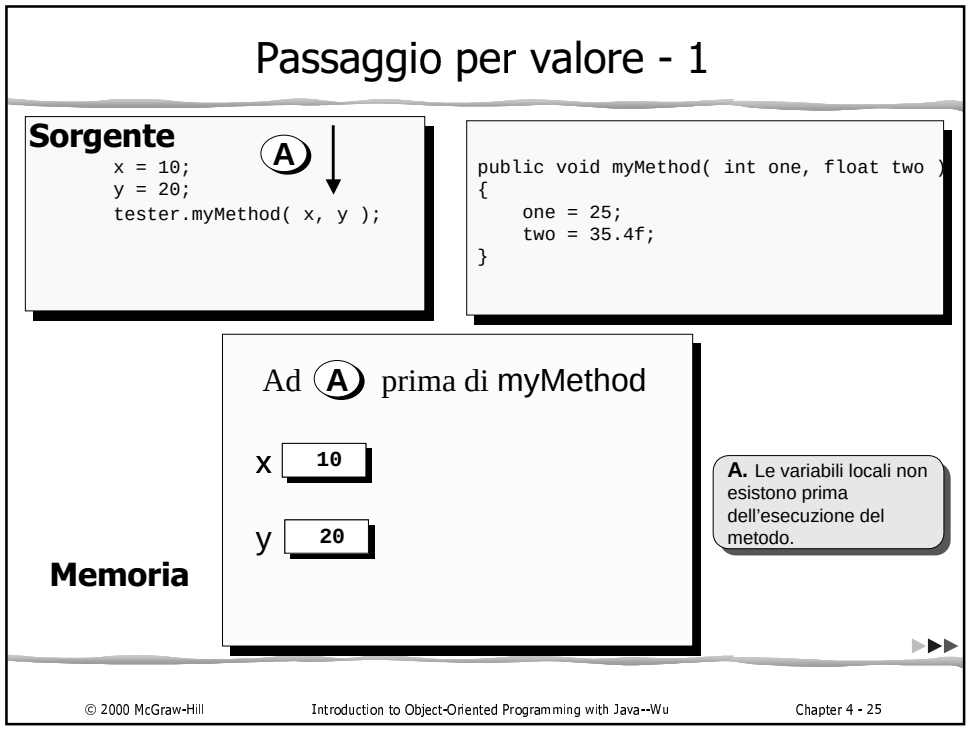
Variabili Locali

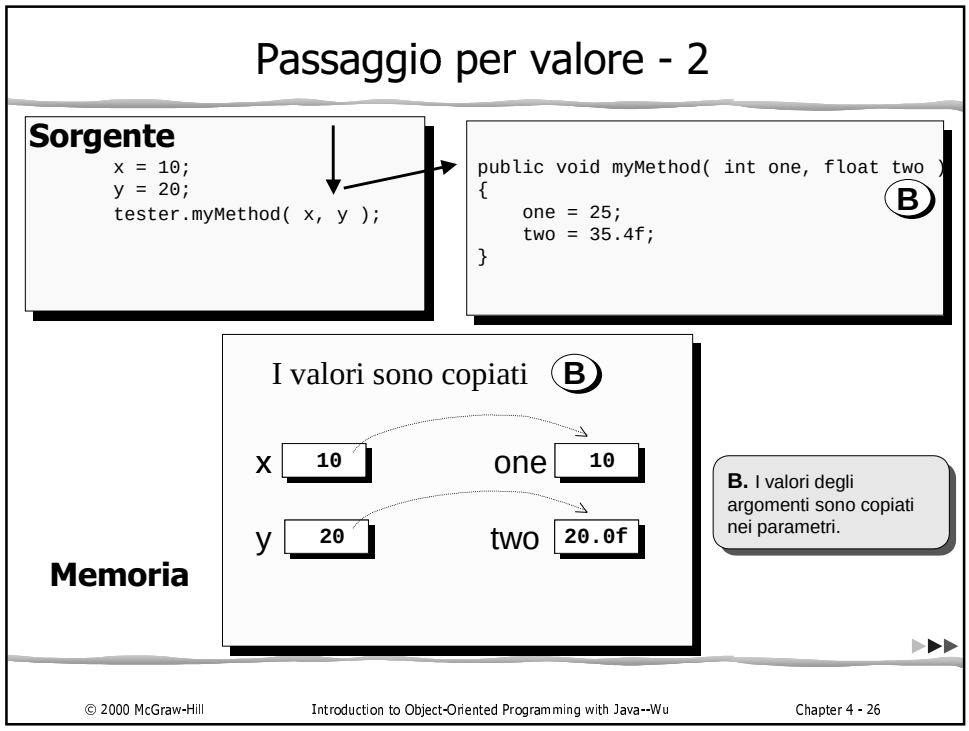

Variabili Locali - 1
Sorgente
A
amt
= yenConverter.fromDollar(200);
public double fromDollar(double dollar)
{
double amount, fee;
fee = exchangeRate - feeRate;
amount = dollar * fee;
return amount;
}
Ad A prima di fromDollar
Memoria
A. Le variabili locali non esistono prima dell'esecuzione del metodo.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 4 - 21

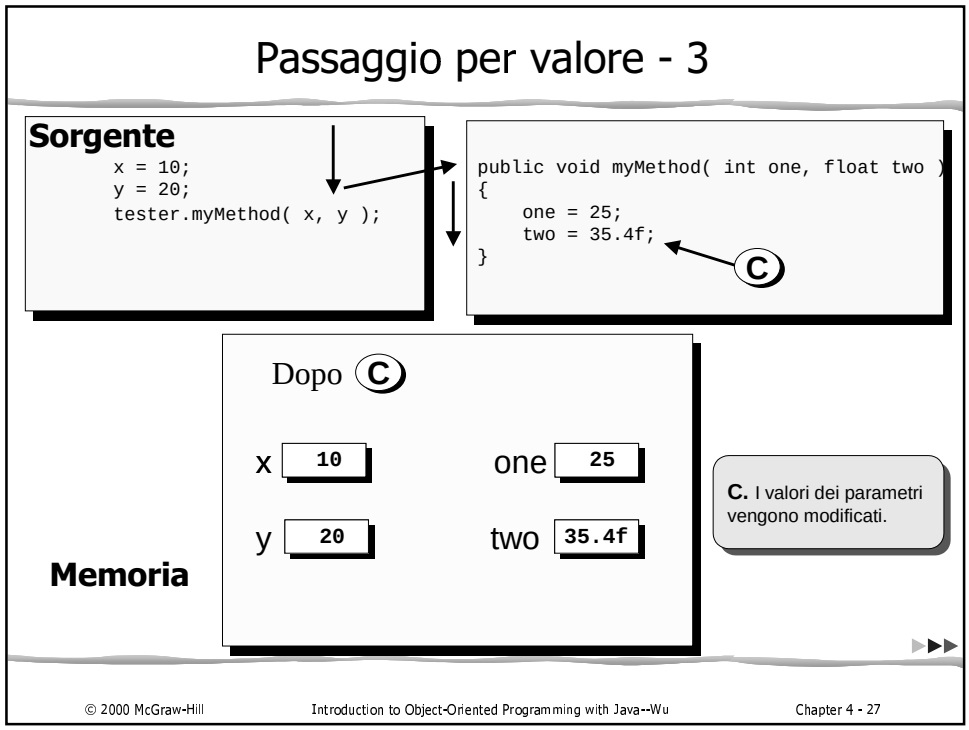

Variabili Locali - 2
Sorgente
amt
= yenConverter.fromDollar(200);
public double fromDollar(double dollar)
{
double amount, fee;
B
fee = exchangeRate - feeRate;
amount = dollar * fee;
return amount;
}
Dopo B
dollar
200.0
amount
fee
Memoria
B. Viene allocato spazio in memoria per le variabili locali e il parametro.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 4 - 22

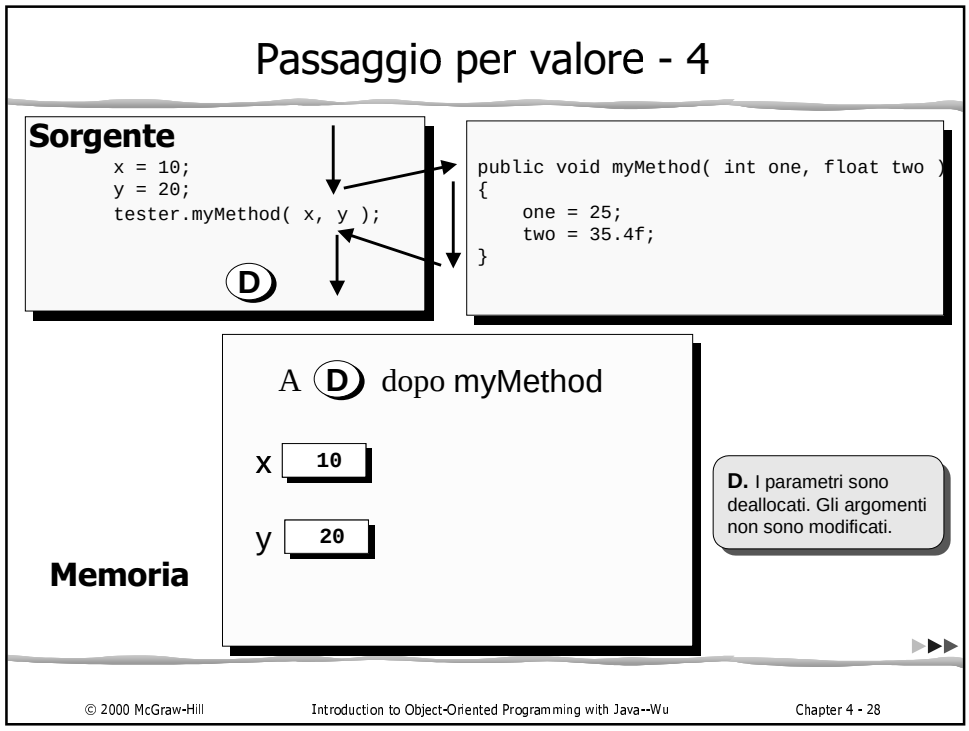

Variabili Locali - 3
Sorgente
amt
= yenConverter.fromDollar(200);
public double fromDollar(double dollar)
{
double amount, fee;
fee = exchangeRate - feeRate;
amount = dollar * fee;
return amount;
}
C
Dopo C
dollar 200.0
amount 24846.3
fee 124.2315
Memoria
C. I valori calcolati sono assegnati alle variabili.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 4 - 23


Variabili Locali - 4
Sorgente
amt
= yenConverter.fromDollar(200);
D
public double fromDollar(double dollar)
{
double amount, fee;
fee = exchangeRate - feeRate;
amount = dollar * fee;
return amount;
}
A D dopo fromDollar
Memoria
D. La memoria viene dellaocata all'uscita di fromDollar.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 4 - 24


Passaggio per valore - 1
Sorgente
x = 10;
y = 20;
tester.myMethod(x, y);
A
public void myMethod(int one, float two)
{
one = 25;
two = 35.4f;
}
Ad A prima di myMethod
x 10
y 20
Memoria
A. Le variabili locali non esistono prima dell'esecuzione del metodo.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 4 - 25


Passaggio per valore - 2
Sorgente
x = 10;
y = 20;
tester.myMethod(x, y);
public void myMethod(int one, float two)
{
one = 25;
two = 35.4f;
}
B
I valori sono copiati B
x 10
y 20
one 10
two 20.0f
Memoria
B. I valori degli argomenti sono copiati nei parametri.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 4 - 26


Passaggio per valore - 3
Sorgente
x = 10;
y = 20;
tester.myMethod(x, y);
public void myMethod(int one, float two)
{
one = 25;
two = 35.4f;
}
C
Dopo C
x 10
y 20
one 25
two 35.4f
Memoria
C. I valori dei parametri vengono modificati.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 4 - 27


Passaggio per valore - 4
Sorgente
x = 10;
y = 20;
tester.myMethod(x, y);
D
public void myMethod(int one, float two)
{
one = 25;
two = 35.4f;
}
A D dopo myMethod
x 10
y 20
Memoria
D. I parametri sono deallocati. Gli argomenti non sono modificati.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 4 - 28

Argomenti & Parametri: Memorandum

1. Gli argomenti sono passati per valore ai metodi.
2. Gli argomenti sono associati ai parametri da destra a sinistra. Il tipo dell'argomento deve essere compatibile con il tipo del parametro.
3. Il numero degli argomenti nella chiamata al metodo deve coincidere con il numero dei parametri nella definizione.
4. I parametri e gli argomenti possono non avere lo stesso nome.
5. Le copie locali, che sono distinte dagli argomenti, sono create anche se i parametri e gli argomenti hanno lo stesso nome.
6. I parametri sono i valori di ingresso di un metodo e sono locali al metodo. I cambiamenti che interessano i parametri non influenzano i valori degli argomenti corrispondenti.

Esempio: Utilizzare una Classe Istanziabile

- Problema

  Scrivere un programma che calcoli i pagamenti mensili e il pagamento totale per un prestito, dato l'ammontare, l'interesse annuale e la durata del prestito.

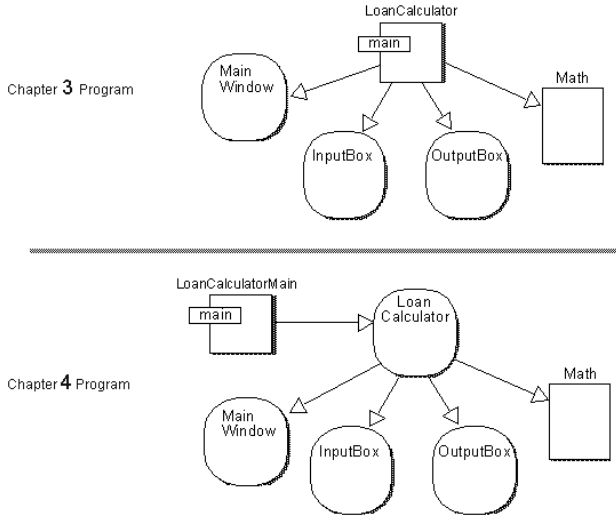
- Compiti principali
  1. Richiedere tre valori in ingresso
  2. Calcolare la rata e la somma delle rate
  3. Visualizzare i risultati

- Formula per il calcolo della rata

$$\text{Monthly Payment} = \frac{L \times R}{\left[1 - \left(\frac{1}{1+R}\right)^{N}\right]}$$

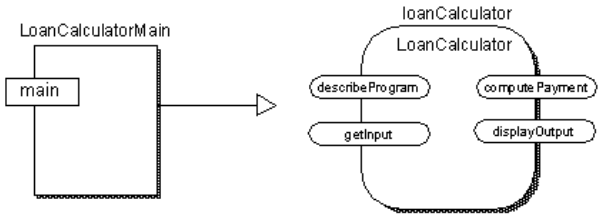
L – prestito
R – tasso mensile
N – numero di pagamenti

Progetto del Programma

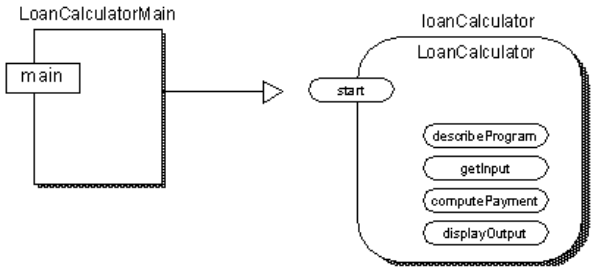


Classe LoanCalculator

Progetto 1



Progetto 2



LoanCalculator – Sviluppo

1. Scrivere la classe main e uno scheletro di LoanCalculator. Tale scheletro include le dichiarazioni degli attributi e un costruttore.
2. Implementare il metodo getInput di LoanCalculator che accetta tre valori di ingresso.
3. Implementare il metodo displayOutput per la visualizzazione dei risultati.
4. Implementare il metodo computePayment per calcolare i pagamenti mensili e totali.
5. Implementare il metodo describeProgram per visualizzare una breve descrizione del programma.