Esempio di classe

Si supponga di lavorare in un’azienda che produce software per la finanza. Vi viene chiesto di progettare e realizzare una classe che rappresenti i conti correnti bancari.

Proviamo a realizzare una classe BankAccount che in modo molto semplice incorpori gli elementi fondamentali di un conto corrente.

Specificare l’interfaccia pubblica di una classe

Comportamento di un conto corrente bancario (astrazione):

- effettuare un deposito
- ritirare un importo
- ottenere il saldo

Specificare l'interfaccia pubblica di una classe: i metodi

Metodi della classe `BankAccount`:

- `deposit`
- `withdraw`
- `getBalance`

Vogliamo che possano funzionare chiamate di metodo del tipo:

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

Specificare l'interfaccia pubblica di una classe: definizione di metodi

- Specificatore di accesso (ad es. `public`)
- Tipo restituito (ad es. `String` o `void`)
- Nome del metodo (ad es. `deposit`)
- Elenco dei parametri (ad es. `double amount` per `deposit`)
- Corpo del metodo in `{ }`

Esempi:

- `public void deposit(double amount) { . . . }`
- `public void withdraw(double amount) { . . . }`
- `public double getBalance() { . . . }`

Sintassi 3.1 Definizione di metodo

```
accessSpecifier returnType methodName(parameterType
parameterName, . . .)
{
   method body
}
```

Esempio:

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

Scopo:

Definire il comportamento di un metodo.

Specificare l'interfaccia pubblica di una classe: definizione del costruttore

- Un costruttore inizializza le variabili di istanza
- Nome del costruttore = nome della classe

```
public BankAccount()
{
   // body--filled in later
}
```

- Il corpo del costruttore viene eseguito quando un nuovo oggetto viene creato
- Le istruzioni nel corpo del costruttore fisserano i valori dei dati interni all'oggetto che viene istanziato
- Tutti i costruttori di una classe hanno lo stesso nome
- Il compilatore può distinguere i costruttori perché essi hanno parametri differenti

Sintassi 3.2 Definizione di costruttore

```
accessSpecifier ClassName(parameterType parameterName, . . .)
{
   constructor body
}
```

Esempio:

```
public BankAccount(double initialBalance)
{
   . . .
}
```

Scopo:

Definire il comportamento di un costruttore.

Interfaccia pubblica di `BankAccount`

I metodi e costruttori pubblici di una classe formano *l'interfaccia pubblica* della classe.

```
public class BankAccount
{
   // Constructors public BankAccount()
   {
      // body--filled in later
   }
   public BankAccount(double initialBalance)
   {
      // body--filled in later
   }
```

Continua

Interfaccia pubblica di BankAccount (continua)

```
    // Methods
    public void deposit(double amount)
    {
        // body--filled in later
    }
    public void withdraw(double amount)
    {
        // body--filled in later
    }
    public double getBalance()
    {
        // body--filled in later
    }
    // private fields--filled in later
}
```

Sintassi 3.3 Definizione di classe

```
accessSpecifier class ClassName
{
   constructors
   methods
   fields
}
```

Esempio:

```
public class BankAccount
{
   public BankAccount(double initialBalance) {. . .}
   public void deposit(double amount) {. . .}
   . . .
}
```

Scopo: Definire una classe, la sua interfaccia pubblica ed i suoi dettagli costruttivi.

Commentare l'interfaccia pubblica

```
/**
   Withdraws money from the bank account.
   @param the amount to withdraw
*/
public void withdraw(double amount)
{
   //implementation filled in later
}
/**
   Gets the current balance of the bank account.
   @return the current balance
*/
public double getBalance()
{
   //implementation filled in later
}
```

Commentare la classe

```
/**
   A bank account has a balance that can be changed by
   deposits and withdrawals.
   */
   public class BankAccount
   {
      . . .
   }
```

- Fornire commenti di documentazione per
  - *ogni classe*
  - *ogni metodo*
  - *ogni parametro*
  - *ogni valore restituito.*

Variabili di istanza (campi di esemplare)

- Un oggetto memorizza i suoi dati in variabili di istanza (campi di esemplare)
- Campo: un termine tecnico per un indirizzo dentro un blocco di memoria
- Istanza di una classe: un oggetto della classe
- La dichiarazione di classe specifica le variabili di istanza

```
public class BankAccount
{
   . . .
   private double balance;
}
```

Variabili di istanza

- Una dichiarazione di variabile di istanza consiste delle seguenti parti:
  - *specificatore di accesso (di norma* `private`*)*
  - *tipo di variabile (ad esempio* `double`*)*
  - *nome della variabile (ad esempio* `balance`*)*
- Ciascun oggetto della classe ha il proprio insieme di dati di istanza
- Tutte le variabili di istanza vanno dichiarate di norma come *private*

Variabili di istanza

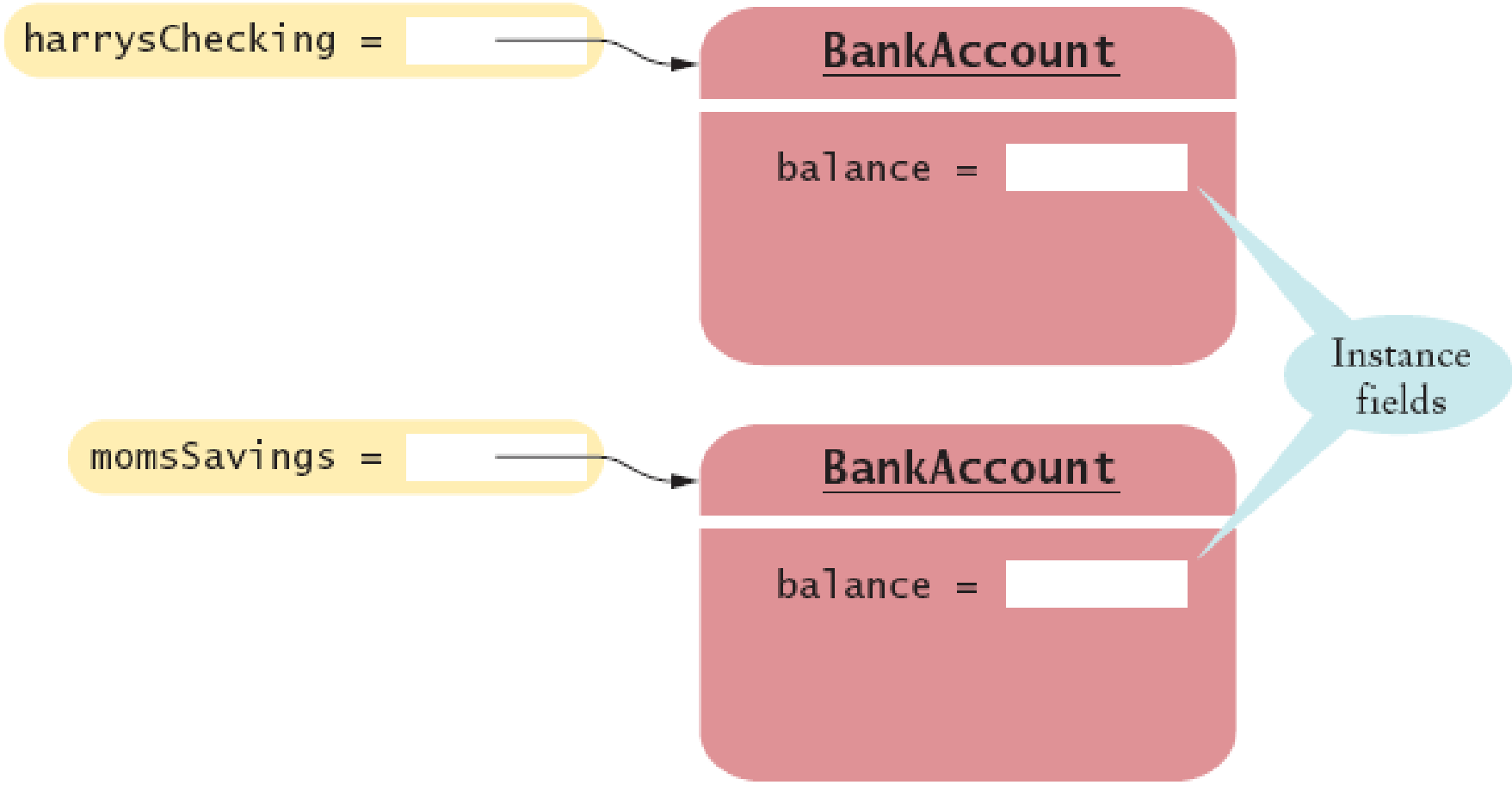

Figure 5 Instance Fields

Sintassi 3.4 Dichiarazione di variabili di istanza

```
accessSpecifier class ClassName
{
   . . .
   accessSpecifier fieldType fieldName;
   . . .
}
```

Esempio:

```
public class BankAccount
{
   . . .
   private double balance;
   . . .
}
```

Scopo:

Definire un campo che è presente in ogni oggetto della classe.

Accedere a dati di istanza

- Il metodo `deposit` della classe BankAccount può accedere il dato di istanza privato, contenuto all'interno della stessa classe:

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

Continua

Accedere a dati di istanza (continua)

- Altri metodi, di classi diverse, non possono:

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

- *Incapsulamento:* è il processo di nascondere i dati dell'oggetto e fornire metodi per l'accesso a tali dati.
- Incapsulare dati significa dichiarare le variabili di istanza come **`private`** e definire metodi **pubblici** che accedono a tali dati.

Realizzare costruttori

- I costruttori contengono istruzioni per inizializzare le variabili di istanza di un oggetto.

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

Esempio di chiamata di un costruttore

- `BankAccount harrysChecking = new BankAccount(1000);`
-
  - *Crea un nuovo oggetto di tipo* `BankAccount`
  - *Chiama il secondo costruttore, poiché viene fornito un parametro)*
  - *Inizializza la variabile parametro* `initialBalance` *a* `1000`
  - *Inizializza il dato di istanza* `balance` *del nuovo oggetto creato al valore contenuto in* `initialBalance`
  - *Restituisce il riferimento ad un oggetto, cioè l'indirizzo di memoria dell'oggetto, come valore dell'espressione* `new`
  - *Memorizza il riferimento all'oggetto nella variabile* `harrysChecking`

Realizzare dei metodi

- Alcuni metodi non restituiscono alcun valore

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

- Alcuni metodi restituiscono un valore di output

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

Esempio di chiamata di un metodo

- `harrysChecking.deposit(500);`
  - *Inizializza la variabile parametro* `amount` *a 500*
  - *Va a prendere il campo* `balance` *dell'oggetto il cui indirizzo è memorizzato in* `harrysChecking`
  - *Aggiunge il valore dell'importo a* `balance` *e memorizza il risultato nella variabile* `newBalance`
  - *Memorizza il valore di* `newBalance` *nel dato di istanza* `balance`*, sovrascrivendo il vecchio valore*

Sintassi 3.5 L'istruzione `return`

```
return expression;
```
o
```
return;
```

Esempio:

```
return balance;
```

Scopo:

Specificare il valore che un metodo restituisce e terminare il metodo immediatamente. Il valore restituito diviene il valore all'interno dell'espressione dove il metodo è stato chiamato.

ch03/account/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:    */
10:    public BankAccount()
11:    {
12:       balance = 0;
13:    }
14:
15:    /**
16:       Constructs a bank account with a given balance.
17:       @param initialBalance the initial balance
18:    */
19:    public BankAccount(double initialBalance)
20:    {
21:       balance = initialBalance;
22:    }
23:
```

Continua

ch03/account/BankAccount.java (cont.)

```
24:    /**
25:       Deposits money into the bank account.
26:       @param amount the amount to deposit
27:    */
28:    public void deposit(double amount)
29:    {
30:       double newBalance = balance + amount;
31:       balance = newBalance;
32:    }
33:
34:    /**
35:       Withdraws money from the bank account.
36:       @param amount the amount to withdraw
37:    */
38:    public void withdraw(double amount)
39:    {
40:       double newBalance = balance - amount;
41:       balance = newBalance;
42:    }
43:
44:    /**
45:       Gets the current balance of the bank account.
46:       @return the current balance
47:    */
```

Continua

ch03/account/BankAccount.java (cont.)

```
48:    public double getBalance()
49:    {
50:       return balance;
51:    }
52:
53:    private double balance;
54: }
```

Categorie di variabili

- Categorie di variabili
  1. *Variabili di istanza (balance in* `BankAccount`*)*
  2. *Variabili locali (*`newBalance` *nel metodo* `deposit`*)*
  3. *Variabili parametro (*`amount` *nel metodo* `deposit`*)*
- Una variabile di istanza appartiene ad un oggetto
- Tali variabili hanno validità fintantoché l'oggetto a cui appartengono viene usato
- In Java, il *garbage collector* periodicamente elimina gli oggetti quando questi non vengono più usati.
- Variabili locali e parametri appartengono ad un metodo
- Le variabili di istanza sono inizializzate ad un valore di default, mentre le variabili locali devono essere inizializzate

Parametri impliciti ed espliciti dei metodi

- Il parametro implicito di un metodo è l'oggetto sul quale il metodo è chiamato
- La parola chiave `this` indica il parametro implicito
- L'uso del nome di una variabile di istanza in un metodo indica la variabile di istanza del parametro implicito.

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

Continua

Parametri impliciti ed espliciti dei metodi (continua)

- `balance` rappresenta il saldo dell'oggetto alla sinistra del punto:

```
momsSavings.withdraw(500)
```

significa

```
double newBalance = momsSavings.balance - amount;
>momsSavings.balance = newBalance;
```

Parametri impliciti e parola chiave `this`

- Ogni metodo ha un parametro implicito
- Il parametro implicito è sempre chiamato `this`
- Eccezione: metodi statici (static) non hanno parametro implicito (si veda il capitolo 8)
- ```
  double newBalance = balance + amount;
  // actually means
  double newBalance = this.balance + amount;
  ```
- Quando ci si riferisce ad una variabile di istanza in un metodo, il compilatore automaticamente gli applica il parametro `this`

```
momsSavings.deposit(500);
```
```

Classi per disegnare figure

Buona pratica: fare una classe per ogni figura da disegnare

```
public class Car
{
   public Car(int x, int y)
   {
   // Remember position
   . . .
   }
   public void draw(Graphics2D g2)
   {
      // Drawing instructions
      . . .
   }
}
```

Disegnare automobili

- Disegnare due automobili: una nell'angolo superiore sinistro della finestra ed un'altra in fondo a destra
- Calcolare la posizione in fondo a destra all'interno del metodo `paintComponent`:

```
int x = getWidth() - 60;
int y = getHeight() - 30;
Car car2 = new Car(x, y);
```

- `getWidth` e `getHeight` sono applicati all'oggetto che esegue `paintComponent`
- Se la finestra viene ridimensionata, il metodo `paintComponent` viene automaticamente chiamato e la posizione dell'auto viene ricalcolata

Continua

Disegnare automobili (continua)

Figure 9
The Car Component Draws Two Car Shapes

Classi di un programma per disegnare automobili

- `Car`: classe responsabile del disegno di una singola automobile
  - *Vengono istanziati due oggetti di tale classe, uno per ogni automobile*
- `CarComponent`: visualizza il disegno
- `CarViewer`: mostra una finestra che contiene `CarComponent`

ch03/car/Car.java

```
01: import java.awt.Graphics2D;
02: import java.awt.Rectangle;
03: import java.awt.geom.Ellipse2D;
04: import java.awt.geom.Line2D;
05: import java.awt.geom.Point2D;
06:
07: /**
08:    A car shape that can be positioned anywhere on the screen.
09: */
10: public class Car
11: {
12:    /**
13:       Constructs a car with a given top left corner
14:       @param x the x coordinate of the top left corner
15:       @param y the y coordinate of the top left corner
16:    */
17:    public Car(int x, int y)
18:    {
19:       xLeft = x;
20:       yTop = y;
21:    }
22:
```

Continua

ch03/car/Car.java (cont.)

```
23:    /**
24:       Draws the car.
25:       @param g2 the graphics context
26:    */
27:    public void draw(Graphics2D g2)
28:    {
29:       Rectangle body
30:             = new Rectangle(xLeft, yTop + 10, 60, 10);
31:       Ellipse2D.Double frontTire
32:             = new Ellipse2D.Double(xLeft + 10, yTop + 20, 10, 10);
33:       Ellipse2D.Double rearTire
34:             = new Ellipse2D.Double(xLeft + 40, yTop + 20, 10, 10);
35:
36:       // The bottom of the front windshield
37:       Point2D.Double r1
38:             = new Point2D.Double(xLeft + 10, yTop + 10);
39:       // The front of the roof
40:       Point2D.Double r2
41:             = new Point2D.Double(xLeft + 20, yTop);
42:       // The rear of the roof
43:       Point2D.Double r3
44:             = new Point2D.Double(xLeft + 40, yTop);
45:       // The bottom of the rear windshield
```

Continua

ch03/car/Car.java (cont.)

```
46:         Point2D.Double r4
47:               = new Point2D.Double(xLeft + 50, yTop + 10);
48:
49:         Line2D.Double frontWindshield
50:               = new Line2D.Double(r1, r2);
51:         Line2D.Double roofTop
52:               = new Line2D.Double(r2, r3);
53:         Line2D.Double rearWindshield
54:               = new Line2D.Double(r3, r4);
55:
56:         g2.draw(body);
57:         g2.draw(frontTire);
58:         g2.draw(rearTire);
59:         g2.draw(frontWindshield);
60:         g2.draw(roofTop);
61:         g2.draw(rearWindshield);
62:     }
63:
64:     private int xLeft;
65:     private int yTop;
66: }
```

ch03/car/CarComponent.java

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import javax.swing.JComponent;
04:
05: /**
06:    This component draws two car shapes.
07: */
08: public class CarComponent extends JComponent
09: {
10:    public void paintComponent(Graphics g)
11:    {
12:       Graphics2D g2 = (Graphics2D) g;
13:
14:       Car car1 = new Car(0, 0);
15:
16:       int x = getWidth() - 60;
17:       int y = getHeight() - 30;
18:
19:       Car car2 = new Car(x, y);
20:
21:       car1.draw(g2);
22:       car2.draw(g2);
23:    }
24: }
```

ch03/car/CarViewer.java

```
01: import javax.swing.JFrame;
02:
03: public class CarViewer
04: {
05:    public static void main(String[] args)
06:    {
07:       JFrame frame = new JFrame();
08:
09:       frame.setSize(300, 400);
10:       frame.setTitle("Two cars");
11:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12:
13:       CarComponent component = new CarComponent();
14:       frame.add(component);
15:
16:       frame.setVisible(true);
17:    }
18: }
19:
```