Costrutto condizionale

Scelte, blocchi

Scelte

- Fino ad ora il corpo dei metodi che abbiamo scritto aveva solo un modo di essere eseguito: in sequenza dalla prima istruzione all'ultima
- In applicazioni non banali il comportamento dei metodi deve variare a seconda dei dati di input: il programma deve prendere delle decisioni in base ai dati in suo possesso
- Vediamo quindi come si fa ad effettuare delle scelte nei programmi

Il costrutto if

- Riconsideriamo la classe **`BankAccount`**
- Il metodo **`withdraw`** preleva dal conto una cifra qualsiasi
- In una applicazione reale, invece, il metodo dovrebbe controllare se la cifra richiesta è disponibile e solo in quel caso effettuare il prelievo
- Per far questo si può usare lo statement (enunciato) **`if`**

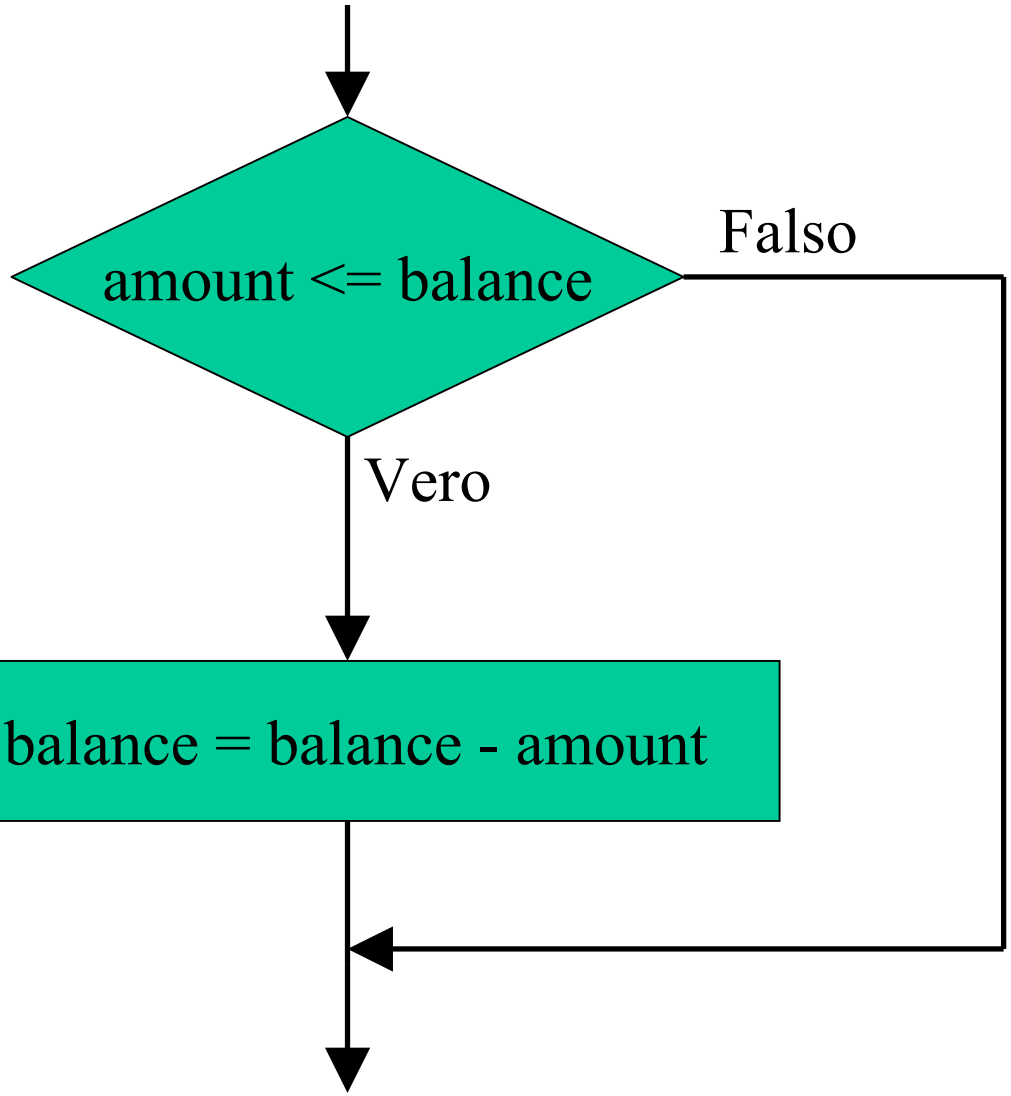
Il costrutto if

```
public class BankAccount {
...
public void withdraw(double amount)
  {
   if (amount <= balance)
     balance = balance - amount;
  }
...
}
```

Il costrutto if

- In questo modo l'assegnamento **`balance = balance - amount`**; viene eseguito solo se la condizione tra parentesi tonde è vera.
- Il significato di **`(amount <= balance)`** è un valore di tipo boolean, cioè un valore nell'insieme {**`true`**, **`false`**}
- L'espressione è vera (**`true`**) se il valore della variabile **`amount`** (parametro esplicito del metodo) è minore o uguale al valore della variabile istanza **`balance`** dell'oggetto si cui si sta eseguendo il metodo

Rappresentazione con diagramma a blocchi



Il costrutto if

- Se la condizione risulta falsa l'**`if`** non fa niente e il controllo passa al comando successivo
- Potremmo pensare, comunque, di avvertire in qualche modo l'utente che il prelievo non è possibile
- Per far questo è possibile aggiungere all'**`if`** un'alternativa (**`else`**) che contiene un comando che viene eseguito solo se la condizione è falsa (**`false`**)

Il costrutto if-else

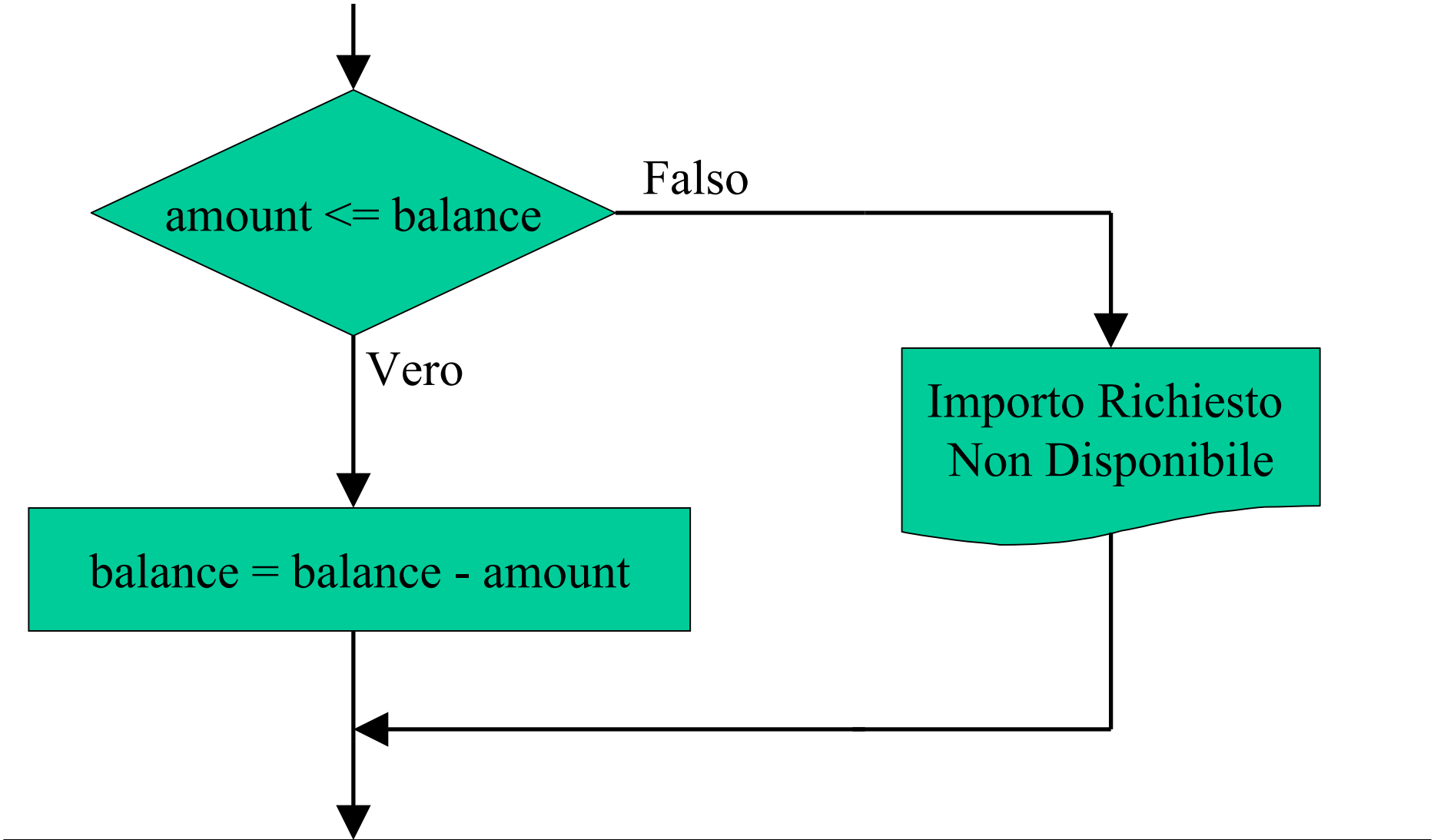
```
public class BankAccount {
...
public void withdraw(double amount) {
   if (amount <= balance)
      balance = balance - amount;
   else
      System.out.println(
     "Importo Richiesto Non Disponibile");
  }
...
}
```

Il costrutto if-else

- La semantica dell'if-else è la seguente:

1. Viene calcolato il valore dell'espressione di tipo **`boolean`** fra parentesi tonde
2. Se l'espressione ha valore **`true`** allora viene eseguito il comando che si trova tra l'espressione e la parola riservata **`else`**.
3. Se l'espressione ha valore **`false`** allora viene eseguito il comando che si trova dopo la parola riservata **`else`**
4. In ogni caso, dopo l'esecuzione del comando corrispondente (caso 2 o caso 3), l'esecuzione continua con l'enunciato che segue

Rappresentazione con diagramma a blocchi

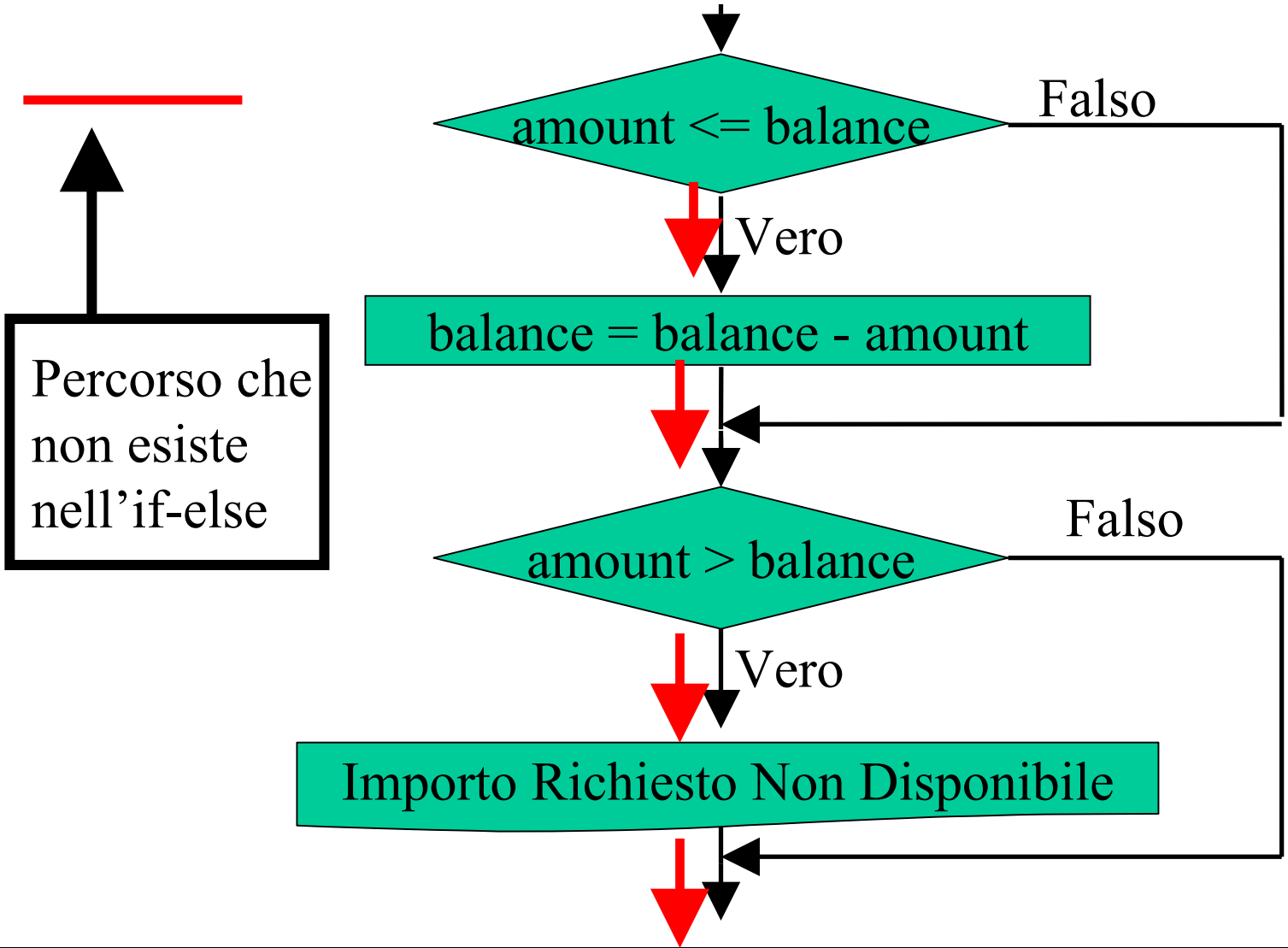


Attenzione: errore!

```
public class BankAccount {
...
public void withdraw(double amount) {
   if (amount <= balance)
     balance = balance – amount;
   if (amount > balance)
     System.out.println(
    “Importo Richiesto Non Disponibile”);
  }
...
}
```

Errore! Non è la stessa cosa dell'if-else: dopo l'esecuzione del primo if il valore di balance può essere diverso e l'espressione negata potrebbe diventare vera, mentre all'inizio era falsa!

Rappresentazione con diagramma a blocchi



Blocchi

- Il costrutto **`if`** o **`if-else`** prevede che sia inserito un solo comando dopo l'espressione tra parentesi e dopo la parola riservata **`else`**
- Invece potremmo aver bisogno di eseguire più di un comando nel "ramo true" e/o nel "ramo false"
- Per far questo usiamo la possibilità, offerta dal linguaggio Java, di creare blocchi di comandi

Blocchi

- Un blocco inizia con { e finisce con }
- All'interno delle parentesi graffe possono essere inseriti zero o più enunciati
- Un blocco può essere inserito in qualunque punto all'interno del corpo di un metodo
- I blocchi possono essere **`annidati`**: un blocco può contenere altri blocchi che a loro volta possono contenere altri blocchi
- **Un blocco viene visto dal compilatore come un unico enunciato**

Blocchi

- Supponiamo ad esempio che se un utente tenta di prelevare del denaro che non ha, oltre alla stampa del messaggio, venga anche decrementato il saldo di una penale costante
- Possiamo scrivere così:

Blocchi

```
public class BankAccount {
...
public static final double PENALE = 5.0;
public void withdraw(double amount) {
  if (amount <= balance) {
    double newBalance = balance - amount;
    balance = newBalance;
  } else {
    System.out.println("Importo Richiesto Non Disponibile");
    balance = balance - PENALE;
    System.out.println("Penale di " + PENALE +
         " euro applicata");
  }
 }
...
}
```

Blocco del ramo "true"

Blocco del ramo "false"

Blocchi e variabili di frame

- Nel blocco del ramo “true” dichiariamo una nuova variabile di tipo double che chiamiamo newBalance
- **In Java, quando si apre un nuovo blocco, viene sempre aggiunto un frame nuovo nella pila di frame contenuta nell’attivazione corrente**
- Le variabili dichiarate all’interno del nuovo blocco hanno scope (contesto) limitato ad esso
- Esse cesseranno di esistere quando l’esecuzione di tutti i comandi del blocco sarà finita e, quindi, il frame verrà buttato via

Visibilità delle variabili e blocchi

- Ricordiamoci che quando viene riferita una variabile con un certo nome all'interno di un metodo la macchina astratta Java segue un metodo ben preciso per rintracciarla:

1. Cerca il nome all'interno del frame in testa alla pila di frame dell'attivazione corrente
2. Se lo trova allora quella è la variabile riferita
3. Se non lo trova lo cerca all'interno dei frame sottostanti dal più recente a quello più in basso.
4. Se non lo trova in nessun frame la variabile non è stata dichiarata (è un errore che viene trovato dal compilatore)

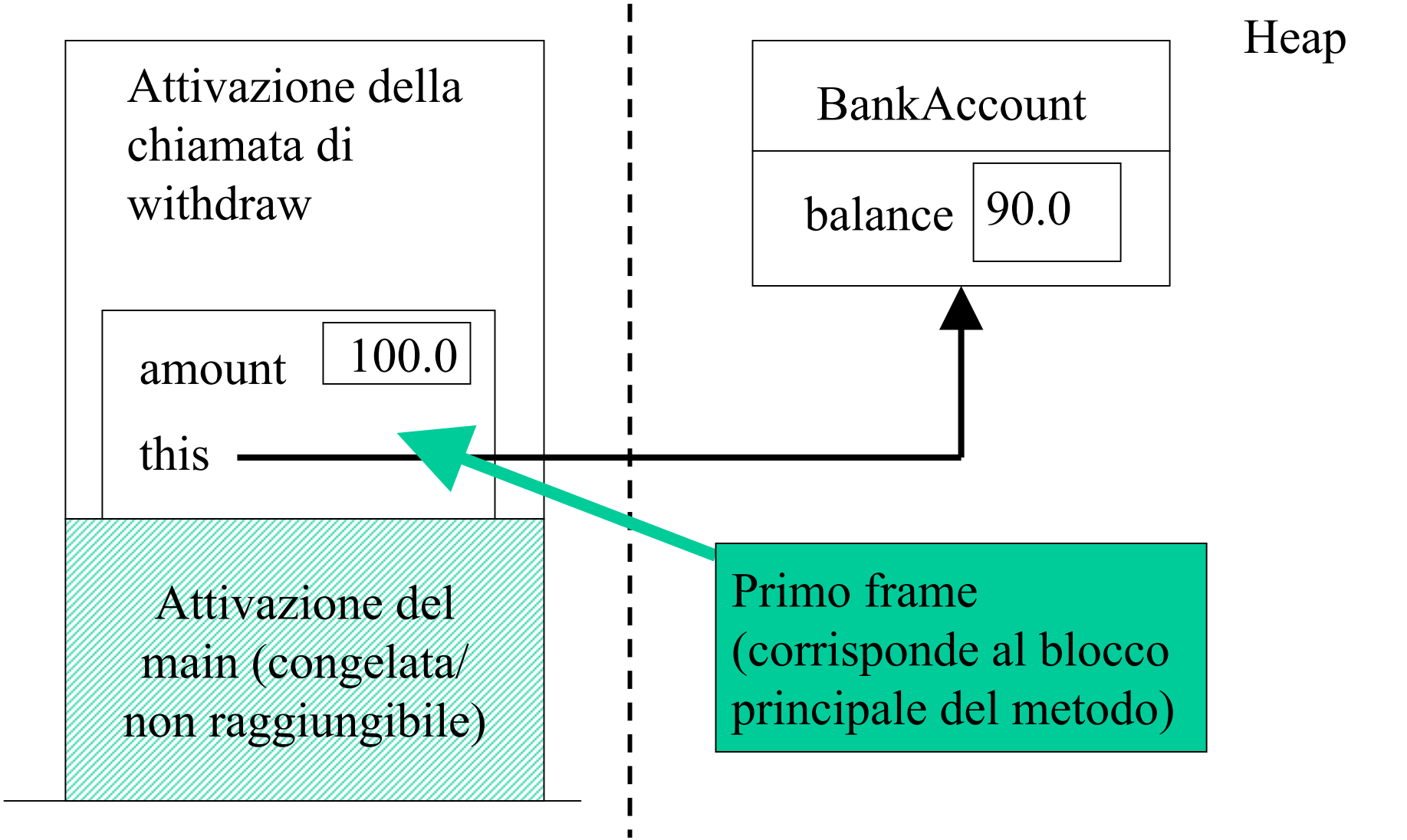
Visibilità delle variabili e blocchi: conflitto

- Il compilatore non permette, all'interno della stessa attivazione, di ridefinire un nome di variabile, anche se in un blocco interno
- Facciamo un esempio. Supponiamo di trovarci nella seguente situazione alla chiamata del metodo withdraw su un certo oggetto creato in un main

Visibilità delle variabili e blocchi


```
public void withdraw(double amount) {
  double newBalance = balance - amount;
  if (newBalance >= 0)
    balance = newBalance;
  else {
    System.out.println("Importo Richiesto Non Disponibile");
    { // Nuovo Blocco
    // non posso ridichiarare newBalance
    double penaltyBalance = balance - PENALE;
    balance = penaltyBalance;
    }
    // penaltyBalance non esiste più!
    System.out.println("Penale di " + PENALE +
          " euro applicata");
  }
 }
```

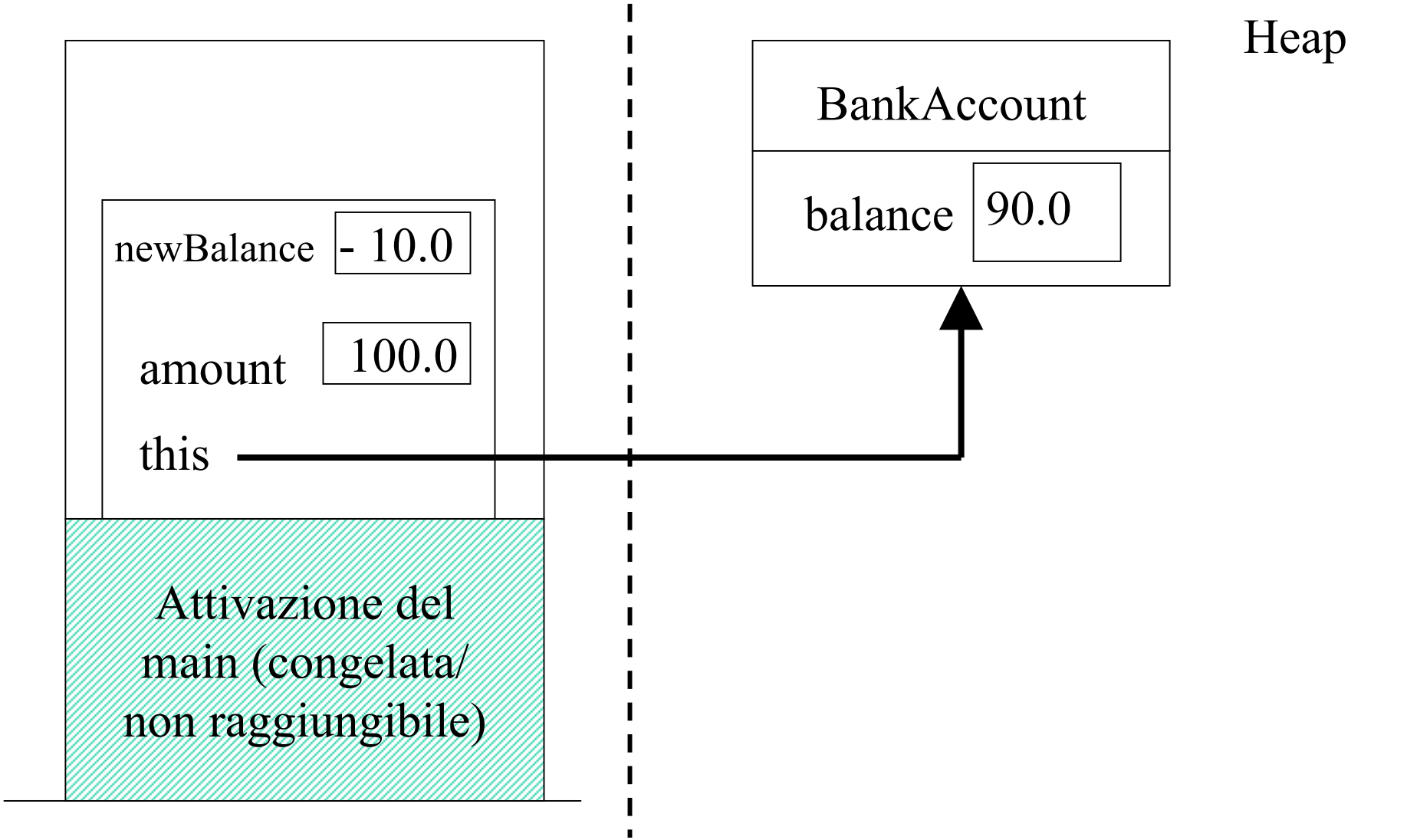
Visibilità delle variabili e blocchi



Visibilità delle variabili e blocchi

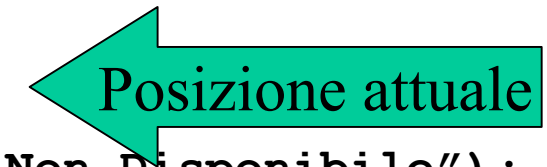
```
public void withdraw(double amount) {
  double newBalance = balance - amount;
  if (newBalance >= 0)
    balance = newBalance;
  else {
    System.out.println("Importo Richiesto Non Disponibile");
    { // Nuovo Blocco
    // Non posso ridichiarare newBalance
    double penaltyBalance = balance - PENALE;
    balance = penaltyBalance;
    } // penaltyBalance non esiste più!
    System.out.println("Penale di " + PENALE +
         " euro applicata");
  }
 }
```

Visibilità delle variabili e blocchi

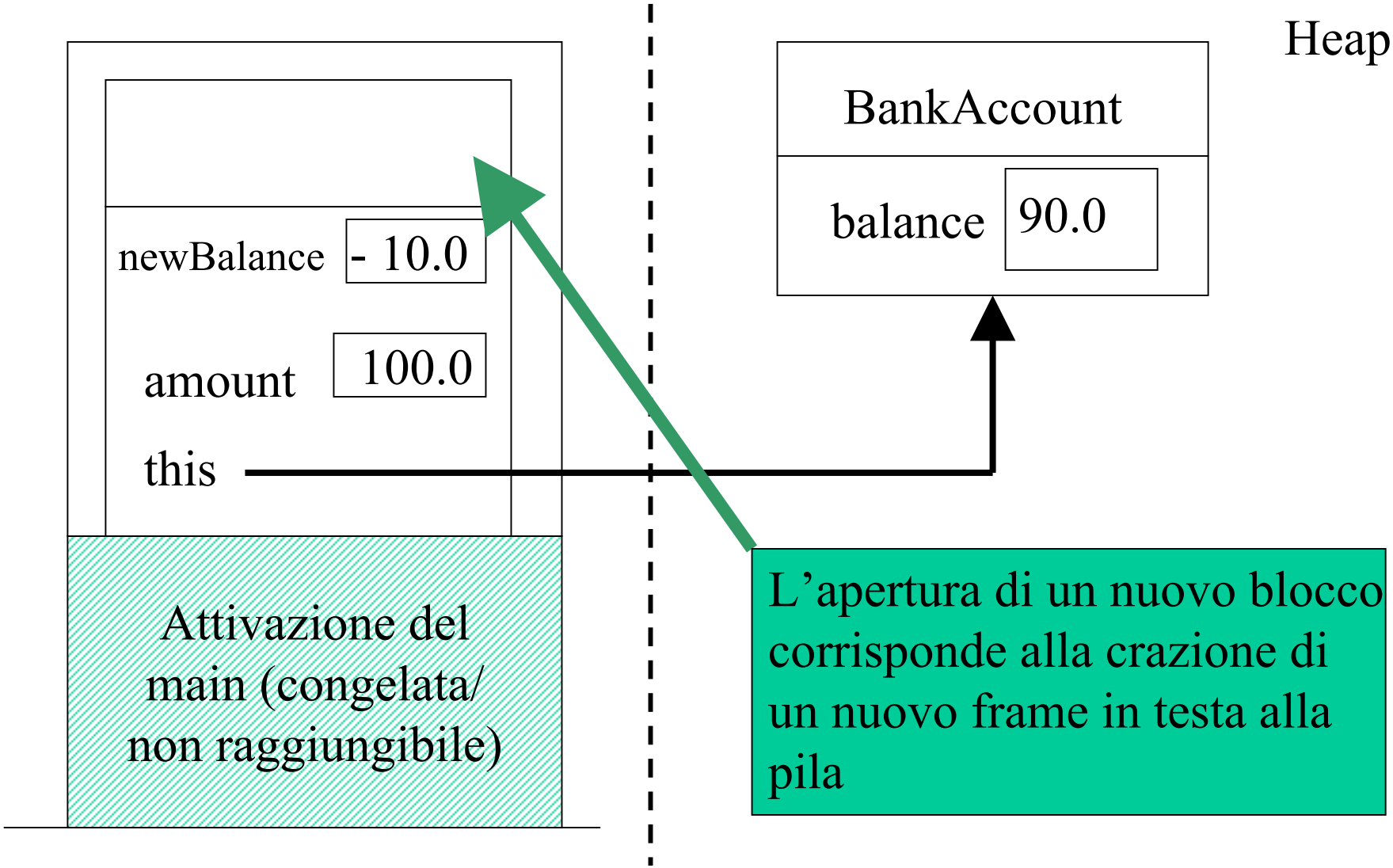


Visibilità delle variabili e blocchi

```
public void withdraw(double amount) {
  double newBalance = balance - amount;
  if (newBalance >= 0)
    balance = newBalance;
  else {
    System.out.println("Importo Richiesto Non Disponibile");
    { // Nuovo Blocco
    // non posso ridichiarare newBalance
    double penaltyBalance = balance - PENALE;
    balance = penaltyBalance;
    }
    // penaltyBalance non esiste più!
    System.out.println("Penale di " + PENALE +
          " euro applicata");
  }
 }
```



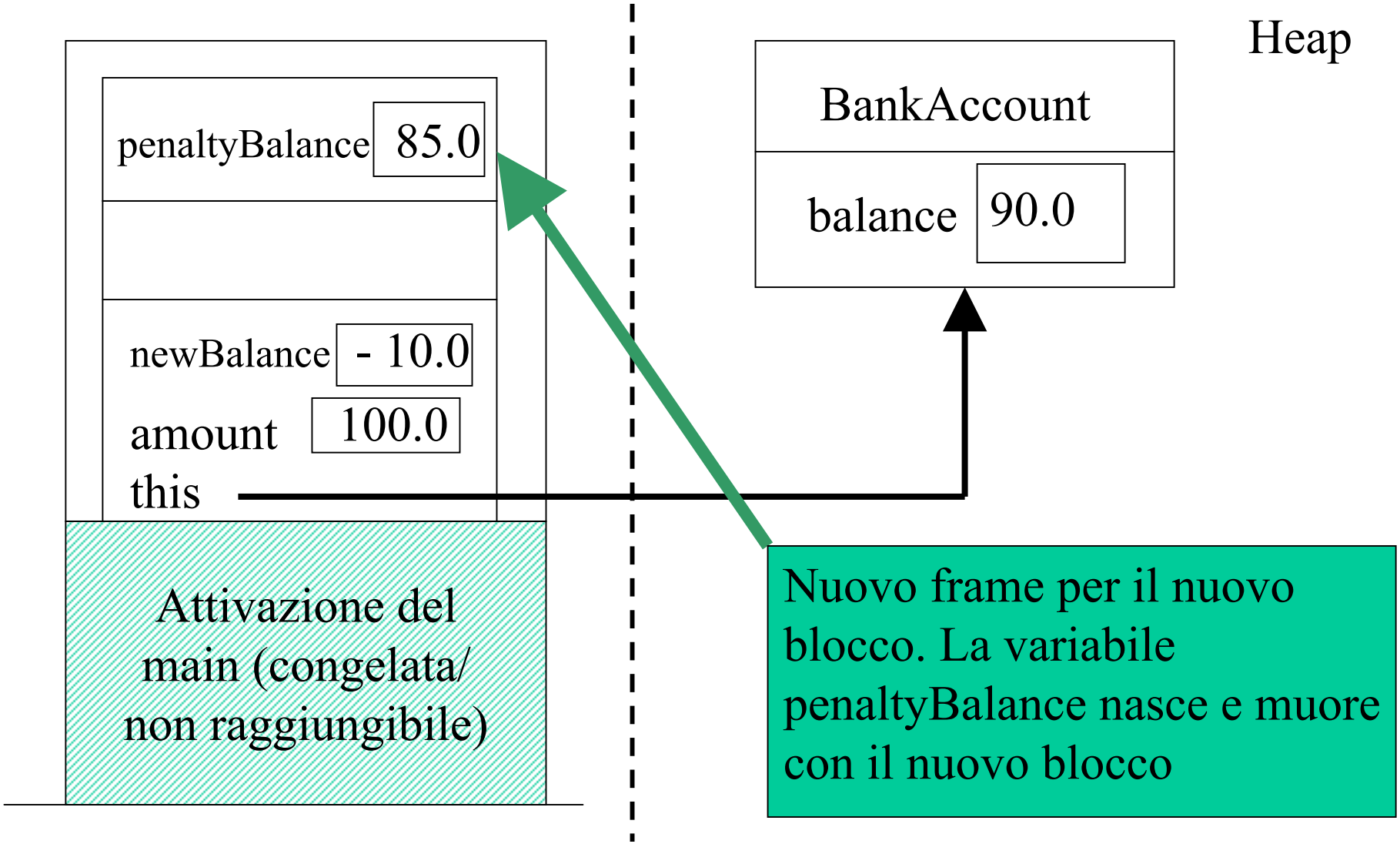
Visibilità delle variabili e blocchi



Visibilità delle variabili e blocchi

```
public void withdraw(double amount) {
  double newBalance = balance - amount;
  if (newBalance >= 0)
    balance = newBalance;
  else {
    System.out.println("Importo Richiesto Non Disponibile");
    { // Nuovo Blocco
    // non posso ridichiarare newBalance
    double penaltyBalance = balance - PENALE;
    balance = penaltyBalance;
    }
    // penaltyBalance non esiste più!
    System.out.println("Penale di " + PENALE +
          " euro applicata");
  }
 }
```

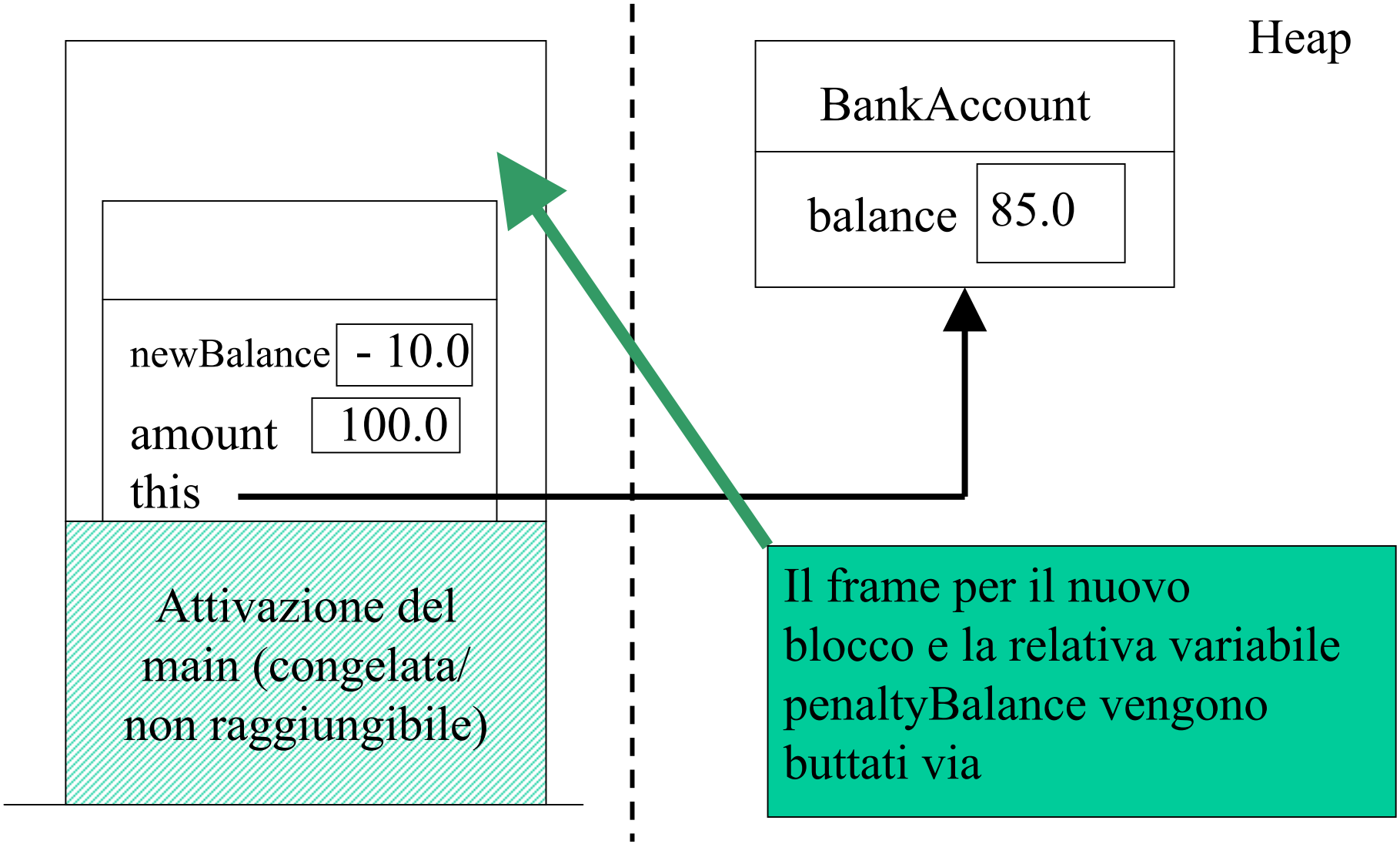
Visibilità delle variabili e blocchi



Visibilità delle variabili e blocchi

```
public void withdraw(double amount) {
  double newBalance = balance - amount;
  if (newBalance >= 0)
    balance = newBalance;
  else {
    System.out.println("Importo Richiesto Non Disponibile");
    { // Nuovo Blocco
    // non posso ridichiarare newBalance
    double penaltyBalance = balance - PENALE;
    balance = penaltyBalance;
    }
    // penaltyBalance non esiste più!
    System.out.println("Penale di " + PENALE +
          " euro applicata");
  }
 }
```

Visibilità delle variabili e blocchi



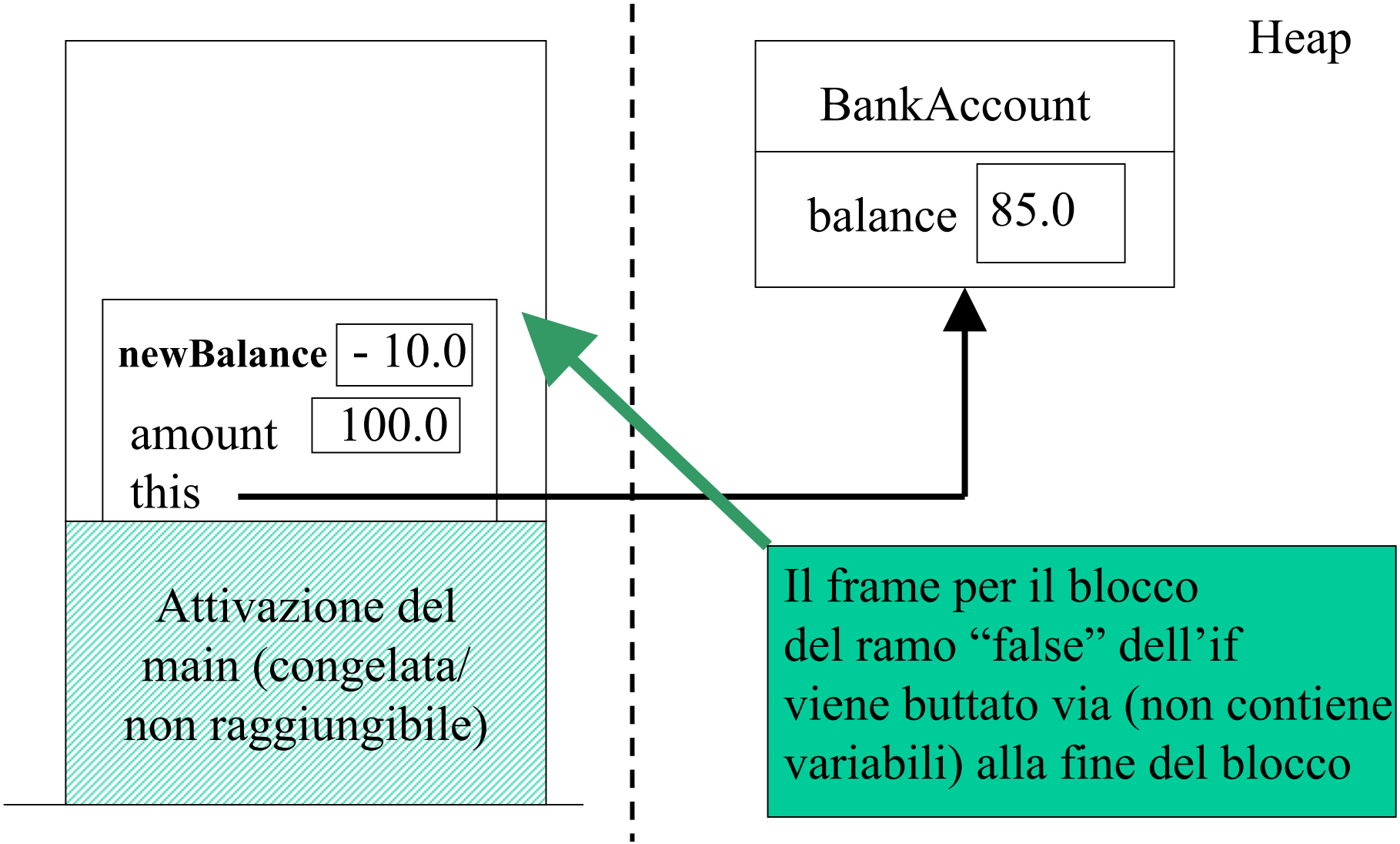
Visibilità delle variabili e blocchi

```
public void withdraw(double amount) {
  double newBalance = balance - amount;
  if (newBalance >= 0)
    balance = newBalance;
  else {
    System.out.println("Importo Richiesto Non Disponibile");
    { // Nuovo Blocco
    // non posso ridichiarare newBalance
    double penaltyBalance = balance - PENALE;
    balance = penaltyBalance;
    }
    // penaltyBalance non esiste più!
    System.out.println("Penale di " + PENALE +
          " euro applicata");
  }
}
```

Posizione attuale

Blocco del ramo "false" chiuso

Visibilità delle variabili e blocchi



Visibilità delle variabili e blocchi

- Alla fine dell'esecuzione del blocco principale di withdraw:

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

- viene cancellato il frame principale, ma anche tutta l'attivazione creata per eseguirlo

Visibilità delle variabili e blocchi