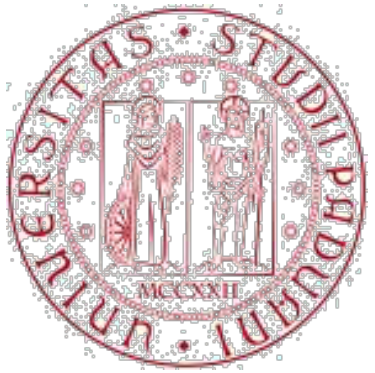
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Parametri e variabili: ripasso e approfondimento



Passaggio di parametri

Accesso alle variabili di esemplare

```
// metodo di altra classe
BankAccount a = new BankAccount(1000);
BankAccount b = new BankAccount();
double money = 500;
a.transfer(b, money);
....
```

```
// classe BankAccount
public void transfer(BankAccount toAccount, double amount)
{   this → this.balance
    balance = balance – amount;
    toAccount.balance += amount;
}
```

Parametri formali ed effettivi

- I parametri espliciti che compaiono ***nell'intestazione*** dei metodi e il parametro implicito ***this*** (usati nella realizzazione dei metodi) si dicono ***Parametri Formali*** del metodo

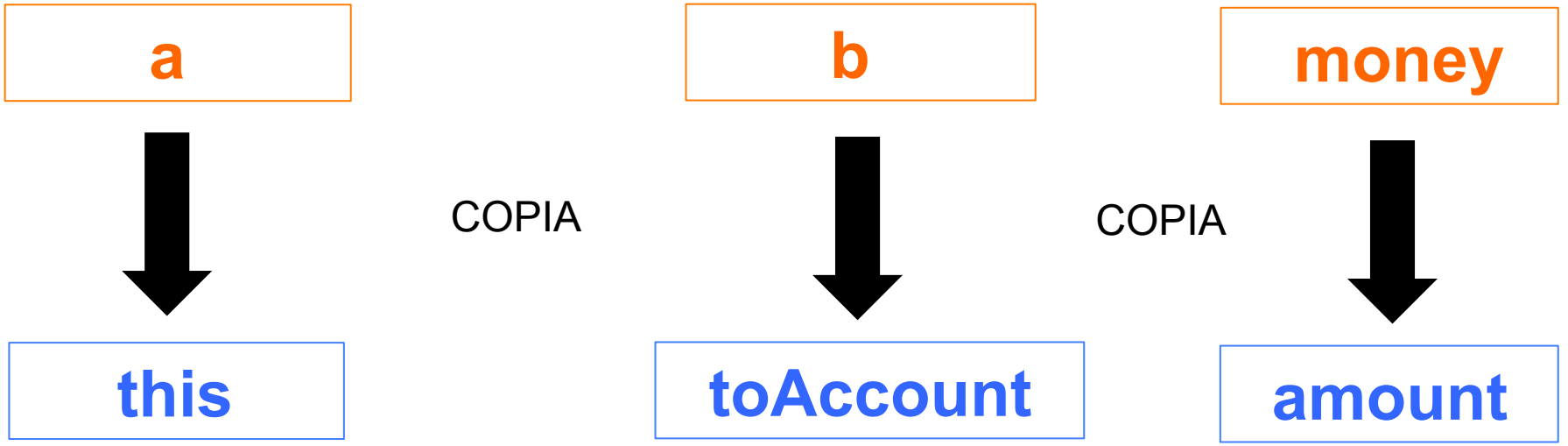
```
public void transfer(BankAccount toAccount, double amount)
{
      this.balance = this. balance – amount;
      toAccount.balance += amount;
}
```

- I parametri forniti ***nell'invocazione*** ai metodi si dicono ***Parametri Effettivi*** del metodo

```
BankAccount a = new BankAccount(1000);
BankAccount b = new BankAccount();
int money = 500;
a.transfer(b, money);
```

Parametri formali ed effettivi

- Al momento dell'esecuzione dell'invocazione del metodo, i **parametri effettivi** sono **copiati** nei **parametri formali**



Il passaggio dei parametri

- **Le "variabili parametro" di un metodo vengono automaticamente definite e inizializzate ogni volta che il metodo viene invocato**
  - Nel metodo invocante
    - l'interprete valuta le espressioni usate come parametri: ciascuna di queste valutazioni genera un valore di un certo tipo (primitivo o oggetto). Ad esempio vede che money contiene 500.
  - Nel metodo invocato
    - PRIMA della sua esecuzione, tali valori vengono usati in normali assegnazioni di valori iniziali per le variabili parametro, che, però, sono operazioni implicite e non figurano esplicitamente nel codice

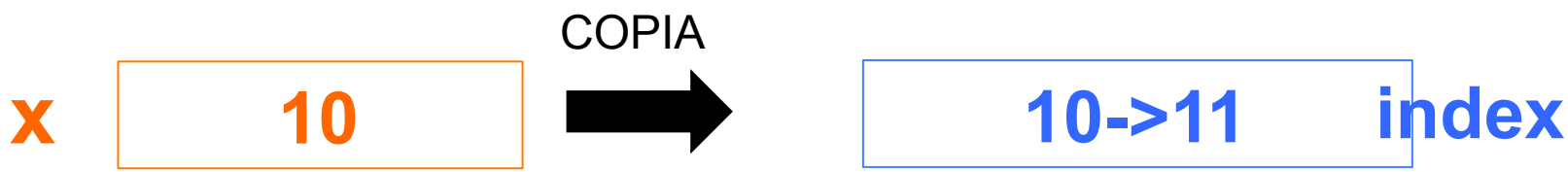
Modificare parametri numerici

- Vogliamo scrivere un metodo **increment** che ha il compito di fornire un nuovo valore per una variabile di tipo numerico

```
public class IncrementaNumero{

    public static void main(String[] args){
        int x = 10;
        increment1(x);
        System.out.println(x);
    }

    public static void increment1(int index) {
        index = index + 1;
    }
}
```

Modificare parametri numerici

```
public class IncrementaNumero{

    public static void main(String[] args){
        int x = 10;
        increment1(x);
        System.out.println(x);
    }

    public static void increment1(int index) {
        index = index + 1;
    }
}
```



invocando il metodo il valore viene copiato nella variabile parametro

in increment1 si modifica questo valore

Modificare parametri numerici

- Come fare?

```
public class IncrementaNumero{

      public static void main(String[] args){
            int x = 10;
            x = increment2(x);
            System.out.println(x);
      }

      public static int increment2(int index) {
            return index + 1;
      }
}
```

Modificare variabili oggetto

- Un metodo può invece modificare lo ***stato*** di un ***oggetto*** passato come parametro (implicito o esplicito)

```
// classe BankAccount // trasferisce denaro dal conto this al conto to
public void transfer(BankAccount toAccount, double amount)
{
    withdraw(amount);              // ritira da un conto
    toAccount.deposit(amount);     // deposita nell'altro conto

} // FUNZIONA !!
```

- Invocando

```
BankAccount a = new BankAccount(10);
BankAccount b = new BankAccount();
a.transfer(b,5);
```

il saldo di a e' 5, saldo di b e' 5

Modificare variabili oggetto

- Ma non può modificare il ***riferimento*** contenuto nella variabile oggetto che ne costituisce il parametro effettivo

```
/ /NON FUNZIONA
public static void swapAccounts(BankAccount x, BankAccount y)
{    BankAccount temp = x;
     x = y;
     y = temp;
}
```

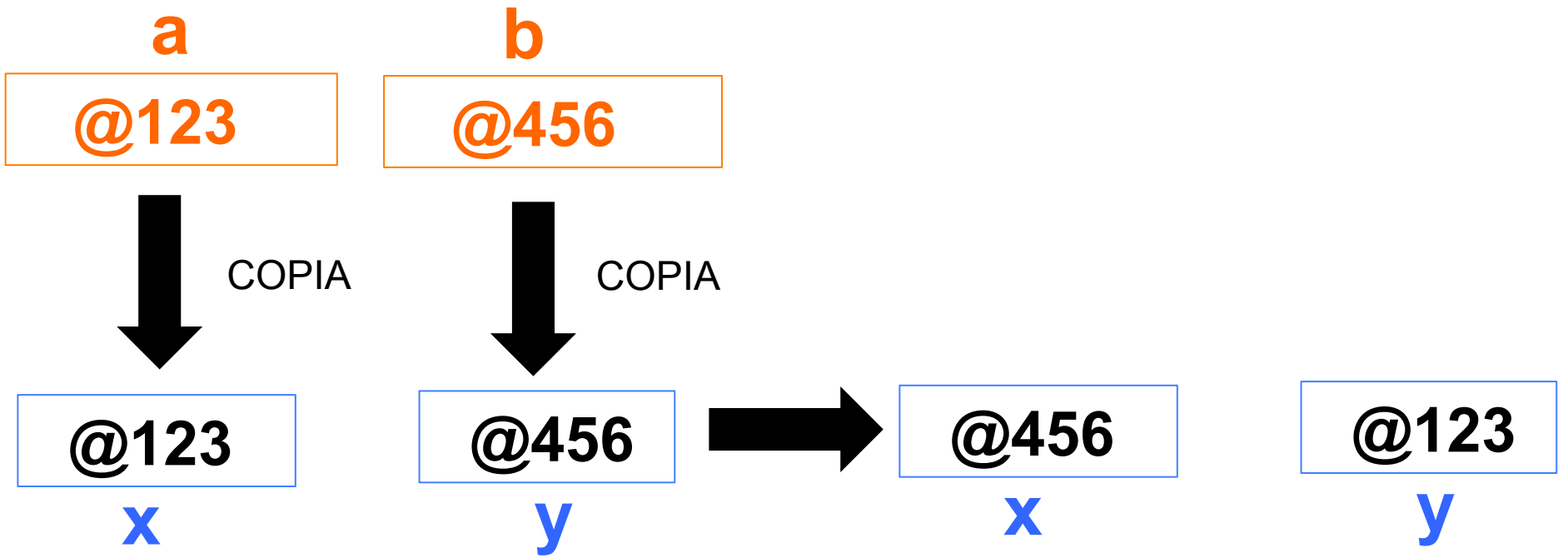
- Invocando

```
BankAccount a = new BankAccount(10);
BankAccount b = new BankAccount();
swapAccounts(a, b);
```

nulla è successo alle variabili a e b

Parametri formali ed effettivi

- Al momento dell'esecuzione dell'invocazione del metodo, i **parametri effettivi** sono **copiati** nei **parametri formali**



Chiamate per valore e per riferimento

- In Java, il passaggio dei parametri è effettuato "per valore", cioè il ***valore*** del parametro effettivo (usato nell'invocazione) viene assegnato al parametro formale (cioè alla variabile parametro)
  - *questo impedisce che il valore del parametro effettivo (nel metodo invocante) possa essere modificato*
- Altri linguaggi di programmazione (come C++) consentono di effettuare il passaggio dei parametri "per riferimento", rendendo possibile la modifica dei parametri effettivi (quando questi sono singole variabili e non espressioni)

Ciclo di vita, inizializzazione e ambito di visibilità di una variabile

Ciclo di vita di una variabile

- In Java esistono quattro diversi tipi di variabili
  - variabili locali (all'interno di un metodo)
  - variabili parametro (dette parametri formali)
  - variabili di esemplare o di istanza
  - variabili statiche o di classe
- Vediamo ora qual è il loro ciclo di vita, cioè quando vengono create e fin quando continuano ad occupare lo spazio in memoria riservato loro

Ciclo di vita: variabili locali

- Una **variabile locale**
  - viene creata quando viene eseguito l'enunciato in cui viene definita
  - viene eliminata quando l'esecuzione del programma esce dal blocco di enunciati in cui la variabile era stata definita
  - se non è definita all'interno di un blocco di enunciati, viene eliminata quando l'esecuzione del programma esce dal metodo in cui la variabile viene definita

Ciclo di vita: variabili parametro formale

- Una **variabile parametro (formale)**
  - viene creata quando viene invocato il metodo
  - viene eliminata quando l'esecuzione del metodo termina

Ciclo di vita: variabili statiche

- Una **variabile statica**
  - viene creata quando la macchina virtuale Java carica la classe per la prima volta
  - viene eliminata quando la classe viene scaricata dalla macchina virtuale Java
    - ai fini pratici, possiamo dire che esiste sempre...

Ciclo di vita: variabili di esemplare

- Una **variabile di esemplare**
  - viene creata quando viene creato l'oggetto a cui appartiene
  - viene eliminata quando l'oggetto viene eliminato
- Un oggetto viene eliminato dalla JVM quando non esiste più alcun riferimento ad esso
  - la zona di memoria riservata all'oggetto viene "riciclata", cioè resa di nuovo libera, dal raccoglitore di rifiuti (**garbage collector**) della JVM, che controlla periodicamente se ci sono oggetti da eliminare

Inizializzazione di una variabile

- Le **variabili di esemplare** e le **variabili statiche**, se non sono inizializzate esplicitamente, vengono **inizializzate automaticamente** ad un valore predefinito
  - **zero** per le variabili di tipo numerico e carattere
  - **false** per le variabili di tipo booleano
  - **null** per le variabili oggetto

Inizializzazione di una variabile

- Le **variabili parametro** vengono inizializzate copiando il valore dei parametri effettivi usati nell'invocazione del metodo

- Le **variabili locali *non*** vengono inizializzate automaticamente, e il compilatore effettua un controllo semantico impedendo che vengano utilizzate prima di aver ricevuto un valore

Ambito di visibilita' di una variabile

- **L'ambito di visibilità** di una variabile indica la parte di codice nel quale è lecito usare la variabile (per leggerne il valore e/o assegnarle un valore)
- Per le **variabili locali** e le **variabili parametro**, l'ambito di visibilità è quello che determina anche il relativo ciclo di vita
- Per le **variabili statiche** e le **variabili di esemplare**, l'ambito di visibilità dipende dalla dichiarazione **public** o **private**

Ambito di visibilita' di variabili statiche o di esemplare

- Se le **variabili statiche** e le **variabili di esemplare** sono dichiarate
  - **public**, sono visibili in ogni parte del programma
  - **private**, sono visibili soltanto all'interno della classe in cui sono definite
- È anche possibile dichiararle **senza indicare uno specificatore di accesso** (accesso di default)
  - sono così visibili anche all'interno di classi che si trovano nello stesso package (cioè di file sorgenti che si trovano nella stessa cartella)
  - Esiste anche lo specificatore protected…

Conflitti tra nomi di variabili

- Conoscere l'**ambito di visibilità** e il **ciclo di vita** di una variabile è molto importante per capire quando e dove è possibile **usare di nuovo il nome di una variabile che è già stato usato**
- Le regole appena viste consentono di usare, in metodi diversi della stessa classe, **variabili locali** o **variabili parametro con gli stessi nomi**, senza creare alcun conflitto, perché
  - i rispettivi ambiti di visibilità non sono sovrapposti
- In questi casi, le variabili definite nuovamente non hanno alcuna relazione con le precedenti