ESERCIZIO

Problema

Scrivere una funzione che scambi due interi

- non opera su oggetti → funzione statica
- scritta dentro a una classe contenitore

Quale intestazione (signature)?

```
public static void scambia(... , ... )
```

Forse questa?

```
public static void scambia(int, int)
```

Ma il passaggio sarebbe per valore!

PASSAGGIO PER RIFERIMENTO

Serve un *passaggio per riferimento.*

Come ottenerlo in Java su un tipo primitivo??

- Java non offre scelta: i tipi primitivi si passano *sempre e solo per valore!*
- Solo gli oggetti si passano per riferimento.

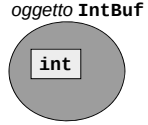
Ma allora...

…basta definire una classe che *incapsuli il tipo primitivo*, e poi passare a scambia *oggetti di quel tipo:*

```
public static void scambia(IntBuf, IntBuf)
```


LA CLASSE `IntBuf`

Definiamo una classe `IntBuf` che *incapsuli il tipo primitivo `int`*

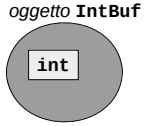


- **Costruttore:** costruisce un oggetto `IntBuf` a partire da un `int`
- **Metodo `getValue`:** recupera il valore `int` da un oggetto `IntBuf`
- **Metodo `setValue`:** cambia il valore `int` contenuto in un oggetto `IntBuf`

```
public class IntBuf {
   private int val;
   public IntBuf(int v) { val = v; }
   public int  getValue() { return val;}
   public void setValue(int v) { val = v;}
}
```

LA FUNZIONE `scambia`

La funzione `scambia` può quindi lavorare su due oggetti `IntBuf`

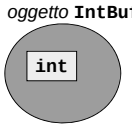


Per scambiare i valori di due oggetti a e b

- si salva in un temporaneo il valore dell'oggetto **a**
- si cambia il valore dell'oggetto **a** rendendolo pari a **b**
- si cambia il valore dell'oggetto **b** rendendolo pari all'intero temporaneo che ospita il vecchio valore di **a**

LA FUNZIONE `scambia`

La funzione `scambia` può quindi lavorare su due oggetti `IntBuf`




```
public class MyLib {
 public static void scambia(IntBuf a, IntBuf b){
  int temp = a.getValue();
  a.setValue(b.getValue());
  b.setValue(temp);
 }
}
```

UN MAIN DI PROVA

```
public class Prova {
 public static void main(String args[]){
  int x = 10, y = 30;
  System.out.println("x, y = " + x + ", " + y);
  IntBuf a = new IntBuf(x), b = new IntBuf(y);
  System.out.println("a, b = " +
          a.getValue() + ", " + b.getValue());
  MyLib.scambia(a,b);
  System.out.println("a, b = " +
          a.getValue() + ", " + b.getValue());
 }
}
```


UN MAIN DI PROVA

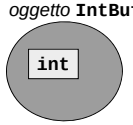
```
public class Prova {
 public static void main(String args[]){
  int x = 10, y = 30;
  System.out.println("x, y = " + x + ", " + y);
  IntBuf a = new IntBuf(x), b = new IntBuf(y);
  System.out.println("a, b = " + a.getValue() + ", "+ b.getValue());
  MyLib.scambia(a,b);
  System.out.println("a, b = " + a.getValue() + ", "+ b.getValue());
 }
}
```

```
C:\temp>java Prova

x, y = 10, 30
a, b = 10, 30
a, b = 30, 10
```

UNA RIFLESSIONE

La classe `IntBuf` prevede un metodo che *modifica il valore:*



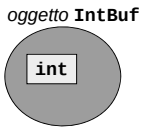
- **Costruttore:** costruisce un oggetto `IntBuf` a partire da un `int`
- **Metodo `getValue`:** recupera il valore `int` da un oggetto `IntBuf`
- **Metodo `setValue`:** cambia il valore `int` contenuto in un oggetto `IntBuf`

```
public class IntBuf {
   private int val;
   public IntBuf(int v) { val = v; }
   public int  getValue() { return val;}
   public void setValue(int v) { val = v;}
}
```

UNA RIFLESSIONE

Dunque:

- **gli oggetti `IntBuf` non sono *valori*, sono *contenitori***
- **il metodo `setValue` appartiene alla categoria dei *trasformatori***




```
public class IntBuf {
 private int val;
 public IntBuf(int v) { val = v; }
 public int  getValue() { return val;}
 public void setValue(int v) { val = v;}
}
```

E LE CLASSI WRAPPER?

Java prevede già *classi standard* per incapsulare valori primitivi: le classi *"wrapper"*

- **esse però sono *valori***
- ***non hanno trasformatori***

Il costruttore costruisce un oggetto "wrapper" a partire da un valore primitivo

Un metodo permette di recuperare il valore primitivo incapsulato in un oggetto "wrapper"

Dunque, le classi *"wrapper"* di Java *non* sono adatte a fungere da contenitori per ottenere il passaggio per riferimento di tipi primitivi.

Servono, in effetti, per *altri scopi.*