Java: lambda expressions

Classe interna anonima

Sottinteso: AnonymousClass implements

```
c.setOnMouseEntered(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
            System.out.print("Entered");
                c.setFill(Color.RED);
    });
```

Sostituzione con una lambda expression

```
c.setOnMouseEntered(new EventHandler<MouseEvent>() {
   public void handle(MouseEvent event) {
         System.out.print("Entered");
            c.setFill(Color.RED);
   });
```

```
c.setOnMouseEntered((MouseEvent event) -> {
         System.out.print("Entered");
         c.setFill(Color.RED);
      });
```

Inferring the functional interfaces

- Does the interface have only one abstract (unimplemented) method?

- Do the parameters (types) of the lambda expression match the parameters (types) of the single method?

- Does the return type of the lambda expression match the return type of the single method?

```
c.setOnMouseEntered((MouseEvent event) -> {
        System.out.print("Entered");
        c.setFill(Color.RED);
      });
```

D: Cosa si aspetta il metodo setOnMouseEntered?

R: Un EventHandler<MouseEvent>

D: L’Interfaccia EventHandler<MouseEvent> ha un solo metodo? Qual'è, e con che firma?

R: si, handle(MouseEvent event)

D: che valore di ritorno restituisce il metodo handle?

R: void

D: La lambda espression è coerente con le attese?

R: si.

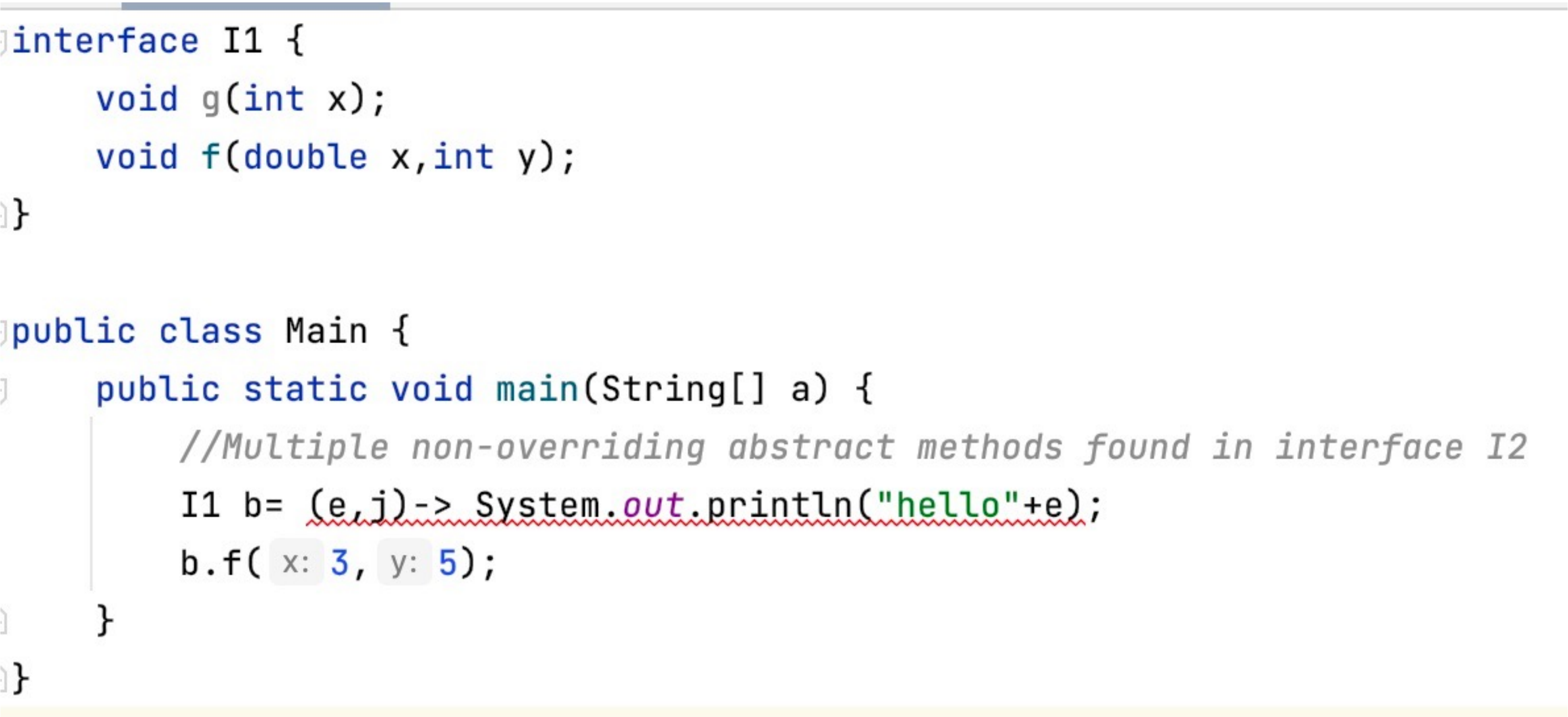
Example

```
public interface I1 {
    int f(int x);
}
```

```
public class Main {
    public static void main(String[] args) {
/*
    I1 a= new I1(){
        @Override
        public int f(int x) {
            return x*3;
        }
    }
*/
    I1 a= (e) -> e*3;

    int c=3+a.f(3);
    System.out.println(c);
    }
}
```

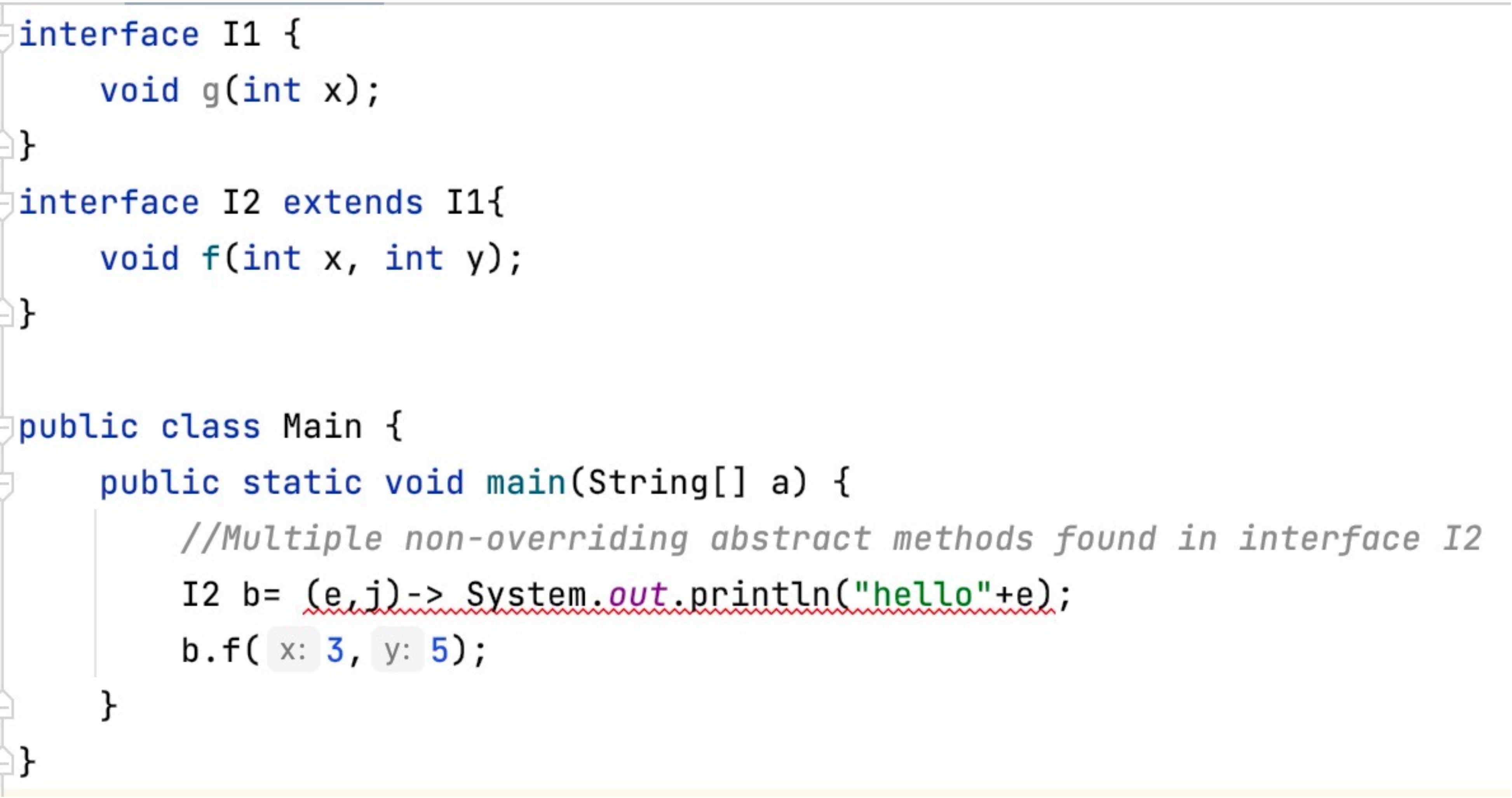
Example



```
interface I1 {
    void g(int x);
    void f(double x,int y);
}

public class Main {
    public static void main(String[] a) {
        //Multiple non-overriding abstract methods found in interface I2
        I1 b= (e,j)-> System.out.println("hello"+e);
        b.f( x: 3, y: 5);
    }
}
```

Example



```
interface I1 {
    void g(int x);
}
interface I2 extends I1{
    void f(int x, int y);
}

public class Main {
    public static void main(String[] a) {
        //Multiple non-overriding abstract methods found in interface I2
        I2 b= (e,j)-> System.out.println("hello"+e);
        b.f( x: 3, y: 5);
    }
}
```

Java: Clonazione

Clonazione

La clonazione...

Ovvero: come costruire una copia

(probabilmente che ritorni true su equals?)

Metodo clone di Object

```
protected Object clone()
            throws CloneNotSupportedException
```

Creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object.

The general intent is that, for any object x,

- the expression: x.clone() != x will be true,
- and that the expression: x.clone().getClass() == x.getClass() will be true, but these are not absolute requirements.
- While it is typically the case that: x.clone().equals(x) will be true, this is not an absolute requirement.

```java
public class Test {
  public static void main(String []a){new Test();}



  Test() {
     P p1=new P();
     p1.x=1;
     p1.y=2;
     P p2=p1;
     P p3=(P)(p1.clone()); // NO! Metodo protected!
     System.out.println(p3);
  }
}
```

```java
class P {
  int x; int y;
  public String toString() {
     return ("x="+x+" ; y="+y);
  }
}
```

clone per la classe P

```
class P implements Cloneable {
…
  public Object clone(){
     try {
      return super.clone();
    } catch (CloneNotSupportedException e) {
      System.err.println("Implementation error");
      System.exit(1);
    }
    return null; //qui non arriva mai  }
  }
}
```

Copia bit a bit

```
public class Test {
  public static void main(String []a){new Test();}
  Test() {
     P p1=new P(); p1.x=5; p1.y=6;
     P p2=p1;
     P p3=p1.clone();
     System.out.println(p1);
     System.out.println(p2);
     System.out.println(p3);
     p1.x=7
     System.out.println(p1);
     System.out.println(p2);
     System.out.println(p3);
  }
}
```

Main di test

```
x=5 ; y=6
x=5 ; y=6
x=5 ; y=6

x=7 ; y=6
x=7 ; y=6
x=5 ; y=6
```

Class V

```
class V  implements Cloneable   {
  int x[];
  V(int s) {
    x=new int[s];
    for (int k=0;k<x.length;k++) x[k]=k;
  }
  public String toString() {
    String s="";
    for (int k:i;) s=s+x[k]+" ";
    return s;
  }
 ... // clone definito come prima
}
```

Main di test

```
public class Test {
  public static void main(String []a){new Test();}

  Test() {
     V p1=new V(5);
     V p2=p1.clone();
     System.out.println(p1);
     System.out.println(p2);
     p1.x[0]=9;
     System.out.println(p1);
     System.out.println(p2);
  }
}
```

```
0 1 2 3 4
0 1 2 3 4

9 1 2 3 4
9 1 2 3 4
```

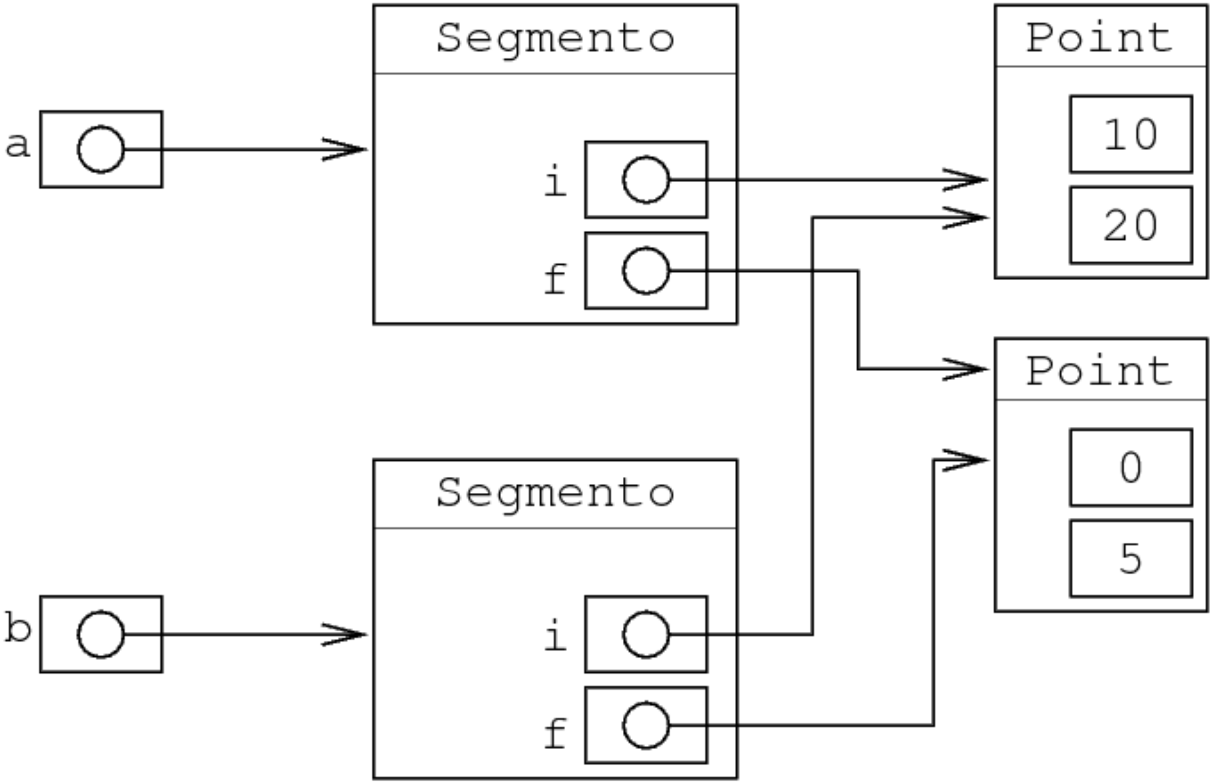
```
class V  implements Cloneable    {
  int x[];V(int s){…}public String toString(){…}
  public Object clone(){
    Object tmp=null;
    try {
      tmp=super.clone();
    } catch (CloneNotSupportedException e) {
      e.printStackTrace(); return null;
    }
    ((V)tmp).x=new int[x.length];
    for (int k:x)((V)tmp).x[k]=x[k];
    return tmp;
  }
}
```

Main di test

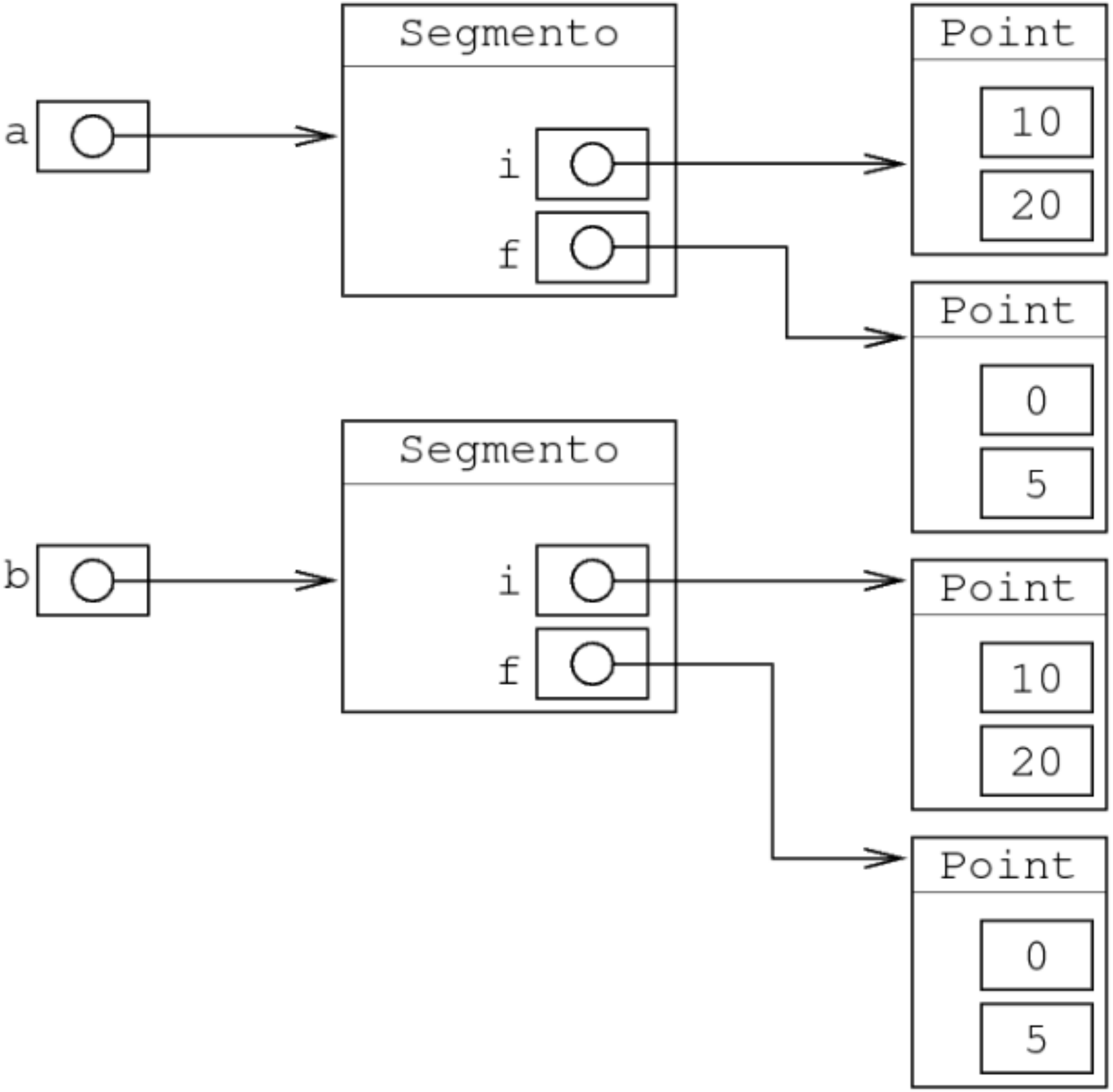
```
public class Test {
  public static void main(String []a){new Test();}
  Test() {
     V p1=new V(5);
     V p2=p1.clone();
     System.out.println(p1);
     System.out.println(p2);
     p1.x[0]=9;
     System.out.println(p1);
     System.out.println(p2)
  }
}
```

```
0 1 2 3 4
0 1 2 3 4
9 1 2 3 4
0 1 2 3 4
```

Copia superficiale (shallow copy)



Copia profonda (deep copy)



Shallow vs. Deep copy

super.clone() (la clone di Object)

Effettua una SHALLOW COPY

Per ottenere una DEEP COPY occorre modificane il risultato.

Ogni volta che ho delle referenze tra le variabili di istanza, devo chiedermi se voglio fare una copia della referenza o dell'oggetto!

C++
Copy Constructor

Versione 1

```
#include <iostream>
using namespace std;
class Line{
   public:
      Line( int len );                  // simple constructor
      ~Line();                          // destructor
};
```

Member functions definitions

```
Line::Line(int len) {
    cout << "Constructor for object at " << this << endl;
}
Line::~Line(void) {
    cout << "Destructor called for " << this << endl;
}
```

main

```
// Main function for the program
int main( ) {
   Line * line = new Line(10);
    cout<<"line : "<<line<<endl;
return 0;
}
```

Allocazione dinamica di memoria (Heap)

```
OUTPUT:
Constructor for object at 0x7f9169c00080
line : 0x7f9169c00080
```

main

```
// Main function for the program
int main( ) {
   Line line2(10);
    cout<<"line2 : "<<&line<<endl;
    return 0;
}
```

Allocazione automatica di memoria (Stack)

```
OUTPUT:
Constructor for object at 0x7ffeeb7b8990
line2 : 0x7ffeeb7b8990
Destructor called for 0x7ffeeb7b8990
```

main

```
void display(Line obj) {
   cout "In display : "<< &obj << endl;
}
```

```
// Main function for the program
int main( ) {
   Line * line = new Line(10);
    cout<<"line : "<<line<<endl;
    display(* line);
return 0;
}
```

Allocazione dinamica di memoria nel main, automatica in display

OUTPUT:

```
Constructor for object at 0x7feefc600000
line : 0x7feefc600000
In display : 0x7ffedfcfb978
Destructor called for 0x7ffedfcfb978
```

main

```
// Main function for the program
int main( ) {
   Line line2(10);
    cout<<"line2 : "<<&line<<endl;
   display(line2);
   return 0;
}
```

Allocazione automatica di memoria

```
OUTPUT:
Constructor for object at 0x7ffee1099990
line2 : 0x7ffee1099990
In display : 0x7ffee1099978
Destructor called for 0x7ffee1099978
Destructor called for 0x7ffee1099990
```

Versione 2

```
#include <iostream>
using namespace std;
class Line{
   public:
      Line( int len );               // simple constructor
      ~Line();                       // destructor
      int getLength( void );
   private:
       int *ptr;
};
```

Member functions definitions

```
Line::Line(int len) {
    cout << "Constructor for object at " << this << endl;
    ptr=new int;
    *ptr=len;
     cout << "Constructor allocating ptr at " << ptr<< endl;
}
Line::~Line(void) {
    cout << ”Destructor called for " << this << endl;
     cout << "Freeing memory! at " << ptr<< endl;
     delete ptr;
}
```

main

```
void display(Line obj) {
   cout "In display : "<< &obj << endl;
}
```

```
// Main function for the progra
int main( ) {
   Line * line = new Line(10);
    cout<<"line : "<<line<<endl;
    display(* line);
    return 0;
}
```

Sembra tutto ok…
Ma c’è un problema.
Dove?

OUTPUT:

```
Constructor for object at 0x7ff5d6500000
Allocating memory for ptr at 0x7ff5d6500010
line : 0x7ff5d6500000
In display : 0x7ffee8590988
Destructor called for 0x7ffee8590988
Freeing memory! at 0x7ff5d6500010
```

main

```
// Main function for the program
int main( )
  Line line2(10);
   cout<<"line2 : "<<&line<<endl;
   display(line2);
   return 0;
}
```

E adesso l'effetto del problema si vede proprio…

OUTPUT:

```
Constructor for object at 0x7ffeeed78990
Allocating memory for ptr at 0x7ff8e6500000
line2 : 0x7ffeeed78990
In display : 0x7ffeeed78978
Destructor called for 0x7ffeeed78978
Freeing memory! at 0x7ff8e6500000
Destructor called for 0x7ffeeed78990
Freeing memory! at 0x7ff8e6500000
copyconstructorexample(76664,0x7fffb7923380) malloc
 *** error for object 0x7ff8e6500000: pointer being
freed was not allocated
```

Come fa una copia degli oggetti il sistema?

Bit a bit…

e come facciamo a sistemarlo?

Con il Copy constructor!

Copy constructor

```
Line::Line(const Line &obj)
{
    cout << "Copy constructor: original at " << &obj <<
                  " copy at " << this << endl;
    ptr = new int;
   *ptr = *obj.ptr; // copy the value
    cout << "original ptr at " << ptr <<
                  " copy of ptr at "<< obj.ptr << endl;
}
```

main

```
// Main function for the program
int main( )
   Line line2(10);
   cout<<"line2 : "<<&line<<endl;
   display(line2);
   return 0;
}
```

Ora è a posto!

OUTPUT:

```
Constructor for object at 0x7ffee9461990
Allocating memory for ptr at 0x7fafd5d00000
line2 : 0x7ffee9461990
Copy constructor: original at 0x7ffee9461990
        copy at 0x7ffee9461978
original ptr at 0x7fafd5d00010
        copy of ptr at 0x7fafd5d00000
In display : 0x7ffee9461978
Destructor called for 0x7ffee9461978
Freeing memory! at 0x7fafd5d00010
Destructor called for 0x7ffee9461990
Freeing memory! at 0x7fafd5d00000
```

In C++…

Quando definite una classe implementatate SEMPRE SUBITO:

- Costruttore
- Distruttore
- Copy constructor!