



Uguaglianza

equals()
e
clone()



Fondamenti di Java

Uguali o identici?



Classe P

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

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

```
    Test() {  
        P p1=new P();  
        p1.x=1;  
        p1.y=2;  
        System.out.println(p1);  
        P p2=p1;  
        p1.x=3;  
        System.out.println(p2);  
    }  
}
```

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

Documentazione 2 - Marco Ronchetti

```

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

    Test() {
        P p1=new P();
        p1.x=1;
        p1.y=2;
        System.out.println(p1);
        P p2=p1;
        p1.x=3;
        System.out.println(p2);
    }
}

```

```

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

```

```

x=1 ; y=2
x=3 ; y=2

```

p1 and p2 refer to te same object!

Fac. Scienza - Università di Trento

6 Programmazione 2 - Marco Ronchetti

```

public class Test {
    public static void main(String a[]) {new Test();}
    Test() {
        P p1=new P();
        p1.x=1; p1.y=2;
        System.out.println(p1);
        P p2=new P();
        p2.x=1; p2.y=2;
        p1.x=3;
        System.out.println(p2);
        System.out.println(p1);
    }
}

```

```

x=1 ; y=2
x=1 ; y=2
x=3 ; y=2

```



Come testare l'eguaglianza?

```
public class Test {  
    public static void main(String a[]){new Test();}  
    Test() {  
        P p1=new P();  
        p1.x=1; p1.y=2;  
        P p2=new P();  
        p2.x=1; p2.y=2;  
        // come testare l'uguaglianza di p1 e p2?  
    }  
}
```



Operatore ==

```
public class Test {  
    public static void main(String[] a){new Test();}  
    Test() {  
        P p1=new P();  
        p1.x=1; p1.y=2;  
        P p2=new P();  
        p2.x=1; p2.y=2;  
        System.out.println(p1==p2);  
    }  
}
```

false



Metodo equals()

```
public class Test {  
    public static void main(String[] a){new Test();}  
    Test() {  
        P p1=new P();  
        p1.x=1; p1.y=2;  
        P p2=new P();  
        p2.x=1; p2.y=2;  
        System.out.println(p1.equals(p2));  
    }  
}
```

false



Metodo equals()

The equals method for class Object implements the **most discriminating** possible equivalence relation on objects; that is, for any reference values x and y, this method **returns true if and only if x and y refer to the same object** (x==y has the value true).

Ma allora a che serve?



equals per la classe P

Equals di Object è la base per implementare il vostro equals

```
class P {
    int x; int y;
    public String toString() {
        return ("x="+x+" ; y="+y);
    }
    public boolean equals(P var){
        return (x==var.x && y==var.y)
    }
}
```



equals() e ==

```
public class Test {
    public static void main(String[] a){new Test();}
    Test() {
        P p1=new P();
        p1.x=1; p1.y=2;
        P p2=new P();
        p2.x=1; p2.y=2;
        System.out.println(p1.equals(p2));
        System.out.println(p1==p2);
    }
}
```

```
true
false
```



Problema 1...

```
public class Test {  
    public static void main(String[] a){new Test();}  
    Test() {  
        P z1=new P();  
        p1.x=1; p1.y=2;  
        P p2=null;  
        System.out.println(p1.equals(p2));  
        System.out.println(p1==p2);  
    }  
}
```

Error!



equals per la classe P, v.2

```
class P {  
    int x; int y;  
    public String toString() {  
        return ("x="+x+" ; y="+y);  
    }  
    public boolean equals(P var){  
        if(var==null) return false;  
        return (x==var.x && y==var.y)  
    }  
}
```



Problema 2...

Equals deve comparare due Objects!

```
public class Test {  
    public static void main(String[] a){new Test();}  
    Test() {  
        P p1=new P();  
        p1.x=1; p1.y=2;  
        Integer p2=new Integer(3);  
        System.out.println(p1.equals(p2));  
        System.out.println(p1==p2);  
    }  
}
```

false

false



equals per la classe P, v.3

```
class P {  
    int x; int y;  
    public String toString() {  
        return ("x="+x+" ; y="+y);  
    }  
    public boolean equals(Object var){  
        if (!(var instanceof P)) return false;  
        if(var==null) return false;  
        return (x==(P)var).x && y==(P)var).y)  
    }  
}
```




Problema 3...

```
public class Test {
    public static void main(String[] a){new Test();}
    Test() {
        P z1=new P();
        p1.x=1; p1.y=2;
        Q p2=new Q();
        p2.x=1; p2.y=2; p2.z=3;
        System.out.println(p1.equals(p2));
        System.out.println(p1==p2);
    }
}
```

```
Class Q extends P {
    int z;
}
```

```
true
false
```



equals per la classe P, v.3b

```
class P {
    int x; int y;
    public String toString() {
        return ("x="+x+" ; y="+y);
    }
    public boolean equals(Object var){
        if (o.getClass() != this.getClass())
            return false;
        if(var==null) return false;
        return (x==(P)var).x && y==(P)var).y)
    }
}
```



e ora...

```
public class Test {
    public static void main(String[] a){new Test();}
    Test() {
        P p1=new P();
        p1.x=1; p1.y=2;
        Q p2=new Q();
        p2.x=1; p2.y=2;
        System.out.println(p1.equals(p2));
        System.out.println(p1==p2);
    }
}
```

```
Class Q extends P {
    int z;
}
```

false

false



Quale soluzione scegliere?

```
if (o.getClass() != this.getClass())
    return false;
```

oppure

```
if (!(var instanceof P)) return false;
```

?

Dipende...



Metodo equals()

The equals method for class Object implements the **most discriminating** possible equivalence relation on objects; that is, for any reference values x and y, this method **returns true if and only if x and y refer to the same object** (x==y has the value true).

```
Object a,b;  
a.equals(b) E' LO STESSO CHE a==b
```

Idem per le sottoclassi di Object, A MENO CHE non
effettuino l'overriding di
public boolean equals(Object x)



Overriding di equals

```
class ... {  
    public boolean equals(Object var){  
        if (o.getClass() != this.getClass())  
            return false;  
        if(var==null) return false;  
        //cicla su tutte le instance variables iv  
        if (this.iv != var.iv) return false;  
        return true;  
    }  
}
```

La Pila in Java equals

```

public class Pila {
    int size;
    int defaultGrowthSize;
    int marker;
    Object contenuto[];
    public boolean equals(Object p) {
        if (!(p instanceof Pila)) return false;
        if (p==null) return false;
        if (marker!=((Pila)p).marker) return false;;
        for (int k=0;k<marker;k++) {
            if (! (contenuto[k].equals(p.contenuto[k]))
                return false;
        }
        return true;
    }
}

```

Non è l'unica semantica possibile!

Proprietà richieste ad equals



The equals method implements an **equivalence relation**:

- It is **reflexive**: for any reference value x, x.equals(x) should return true.
- It is **symmetric**: for any reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
- It is **transitive**: for any reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.



Proprietà richieste ad equals

Additional properties:

- **It is consistent**: for any reference values *x* and *y*, multiple invocations of *x.equals(y)* consistently return true or consistently return false, provided no information used in equals comparisons on the object is modified.
- For any non-null reference value *x*, ***x.equals(null)*** should return false.



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.

```
public class Test {
    public static void main(String []a){new Test();}
    Test() {
        P p1=new P();
        p1.x=1;
        p1.y=2;

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

        P p2=p1.clone(); // NO! Metodo protected!
        System.out.println(p2);
    }
}
```



clone per la classe P

```
class P implements Cloneable {  
...  
    public Object clone(Object var){  
        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=1;  
        p1.y=2;  
        P p2=p1.clone();  
        p1.y=9;  
        System.out.println(p2);  
        System.out.println(p1);  
    }  
}
```

x=1 ; y=2
x=9 ; y=2



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=0;k<x.length;k++) 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[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; //l'oggetto che creero'
        try {
            tmp=super.clone(); //copia di base
        } catch (CloneNotSupportedException e) {
            e.printStackTrace(); return null;
        }
        ((V)tmp).x=new int[x.length]; //adattamento
        for (int k=0;k<x.length;k++) ((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[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



Shallow vs. Deep copy

`super.clone()`

Effettua una **SHALLOW COPY**

Per ottenere una **DEEP COPY** occorre modificare 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!



clone per la Pila

Esercizio!

Attenzione: gestire il caso che gli oggetti
contenuti nella Pila non implementino
Cloneable