

1

# Uguaglianza e Identità

(no, non avete sbagliato corso...)

# Fondamenti di Java



Che vuol dire "uguaglianza"?

Che vuol dire "Identità"?

Che differenza c'è?

# Uguaglianza

```
public class Test {  
    public static void main(String[] a){ new Test(); }  
    Test() {  
        int k1 = 1;  
        int k2 = 1;  
        System.out.println(k1==k2) ;  
    }  
}
```

true

# Uguaglianza

```
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;  
    }  
}
```

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

p1 e p2 sono uguali?

# Uguaglianza

```
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**

# Uguaglianza

```
public class Test {  
    public static void main(String[] a){new Test();}  
    Test() {  
        int k1 = 1;  
        int k2 = k1;  
        System.out.println(k1==k2) ;  
    }  
}
```

true

# Uguaglianza

```
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;  
        System.out.println(p1==p2);  
    }  
}
```

true

# Uguaglianza

```
int k1=1;  
int k2=1;
```

k1==k2 ?     **TRUE**

```
int k1=1;  
int k2=k1;
```

k1==k2 ?     **TRUE**

```
P p1=new P();  
p1.x=1; p1.y=2;  
P p2=new P();  
p2.x=1; p2.y=2;
```

p1==p2 ?     **FALSE**

```
P p1=new P();  
p1.x=1; p1.y=2;  
P p2= p1;
```

p1==p2 ?     **TRUE**

Perché? (ricordiamoci dell'allocazione di memoria...)



# Oggetti diversi

```
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;  
        System.out.println(p2);  
        p1.x=3;  
        System.out.println(p1);  
        System.out.println(p2);  
    }  
}
```

x=1 ; y=2

x=1 ; y=2

x=3 ; y=2

x=1 ; y=2

# Oggetti diversi?

```
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;  
        p2.x=3;  
        System.out.println(p1);  
    }  
}
```

x=1 ; y=2

x=3 ; y=2

**p1** e **p2** si riferiscono allo **stesso** oggetto!

# 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);  
    }  
}
```

== testa l'**identità**  
(due riferimenti puntano  
allo stesso oggetto)

**false**

java.lang

## Class Object

java.lang.Object

```
public class Object
```

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

See Also:

Class

### Constructor Summary

#### Constructors

##### Constructor and Description

`Object()`

### Method Summary

#### Methods

##### Modifier and Type

protected `Object`

boolean

##### Method and Description

`clone()`

Creates and returns a copy of this object.

`equals(Object obj)`

Indicates whether some other object is "equal to" this one.

`equals()` = testa l'**uguaglianza**  
(due riferimenti puntano a oggetti «uguali»)

# 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 cosa serve?!?

# Il “nostro” 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)  
    }  
}
```

`Object.equals ()` è la base per ridefinirne  
il comportamento  
secondo quanto necessario per l'applicazione




# 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 p1=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)  
    }  
}
```

java.lang

## Class Object

java.lang.Object

public class Object

Class Object is the root

Since:

JDK1.0

See Also:

Class

Ma abbiamo fatto overriding o overloading?

```
class P {  
    public boolean equals(P var) ...  
    ...  
}
```

## Constructor Summary

### Constructors

#### Constructor and Description

`Object()`

## Method Summary

### Methods

#### Modifier and Type

protected `Object`

#### Method and Description

`clone()`

Creates and returns a copy of this object.

boolean

`equals(Object obj)`

Indicates whether some other object is "equal to" this one.

java.lang

## Class Object

java.lang.Object

public class Object

Class Object is the root

Since:

JDK1.0

See Also:

Class

### Constructor Summary

#### Constructors

Constructor and De

Object()

### Method Summary

#### Methods

Modifier and Type

protected Object

boolean

clone()

Creates and returns a copy of this object.

equals(Object obj)

Indicates whether some other object is "equal to" this one.

Ma abbiamo fatto overriding o overloading?

```
class P {  
    public boolean equals(P var) ...  
...}
```

che succede se

```
P p1=new P();
```

```
p1.x=1; p1.y=2;
```

```
Integer p2=new Integer(3);
```

```
System.out.println(p1.equals(p2));
```

# 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==null) return false;  
        if (!(var instanceof P)) 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 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;  
}
```


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(var==null) return false;  
        if (var.getClass() != this.getClass())  
            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 z1=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...

# Uguaglianza e sottoclassi.

- Per verificare la compatibilità del tipo dell'oggetto `o` passato come parametro a `equals()` abbiamo due possibilità

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

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

- La prima vincola il tipo del parametro `o` a coincidere con quello dell'oggetto su cui `equals()` è invocato
- La seconda consente il confronto anche con oggetti appartenenti a una sua sottoclasse
- **Quale usare? Dipende dall'applicazione!**

# 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.

# E le sottoclassi?

```
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;  
}
```

```
true  
false
```

# E le sottoclassi?

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



# E le sottoclassi?

```
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

# E le sottoclassi?

- Per verificare la compatibilità del tipo dell'oggetto `o` passato come parametro a `equals()` abbiamo due possibilità

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

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

- La prima vincola il tipo del parametro `o` a coincidere con quello dell'oggetto su cui `equals()` è invocato
- La seconda consente il confronto anche con oggetti appartenenti a una sua sottoclasse
- **Quale usare? Dipende dall'applicazione!**

# Proprietà di `equals()`

- Il metodo `equals()` implementa una relazione di equivalenza fra elementi non nulli e deve soddisfare le seguenti proprietà:
  1. **riflessiva**: per ogni riferimento non nullo `x`, `x.equals(x)` ritorna `true`
  2. **simmetrica**: per ogni riferimento non nullo `x` e `y`, `x.equals(y)` ritorna `true` se e solo se `y.equals(x)` ritorna `true`
  3. **transitiva**: per ogni riferimento non nullo `x`, `y`, `z`, se `x.equals(y)` e `y.equals(z)` ritornano `true` allora `x.equals(z)` ritorna `true`
  4. **consistente**: per ogni riferimento non nullo `x` e `y`, invocazioni diverse di `x.equals(y)` ritornano lo stesso valore, se nessuna delle informazioni usate da `equals()` sono state modificate
  5. per ogni riferimento non nullo `x`, `x.equals(null)` ritorna `false`

Se il metodo viene ridefinito, garantire queste proprietà  
**è compito del programmatore**

## Ridefinire `equals ()` non basta...

- La classe `Object` fornisce anche un metodo `hashCode ()`
- Rappresenta una «funzione *hash*» non iniettiva (e quindi non invertibile)  
**che mappa un oggetto su un intero**
  - Sono possibili «collisioni» cioè oggetti diversi mappati sullo stesso intero
- Viene utilizzata dal Java runtime per gestire in maniera efficiente strutture dati di uso comune
- Il comportamento di `hashCode ()` è legato al metodo `equals ()`