Tipi primitivi

Il linguaggio Java offre alcuni tipi di dato primitivi

Una variabile di tipo primitivo può essere utilizzata direttamente. Non è un riferimento e non ha senso tentare di istanziarla mediante l'operatore new.

La variabile può essere quindi utilizzata direttamente dopo la sua dichiarazione

Tipo primitivo	Descrizione
boolean	valori che possono essere true e false
char	caratteri sono di 16 bit e codificati Unicode
byte	interi di 8 bit con segno e codifica in complemento a due
short	interi di 16 bit con segno e codifica in complemento a due
int	interi di 32 bit con segno e codifica in complemento a due
long	interi di 64 bit con segno e codifica in complemento a due
float	reali di 32 bit in virgola mobile (IEEE 754)
double	reali di 64 bit in virgola mobile (IEEE 754)

Tipi primitivi

Ad **esempio**, il codice seguente dichiara una variabile di tipo intero, le assegna il valore 5 e stampa a schermo il suo contenuto:

```
public static void main(String args[]){
        int x;
        x = 5;
        System.out.println("X = "+x);
}
```

Invece, il codice seguente è completamente **sbagliato**. int è un tipo di dato primitivo ….

```
public static void main(String args[]){
        int x = new int();
        x = 5;
        System.out.println("X = "+x);
}
```

Riferimenti

Le classi sono utilizzate per modellare nuovi tipi di dato astratto: il nome della classe rappresenta un nuovo tipo di dato

Una variabile che referenzia un oggetto è un riferimento. Non contiene il valore dell'oggetto ma solo l'indirizzo di memoria a cui è allocato quest'ultimo

Un riferimento appena creato è automaticamente inizializzato con il valore null; ogni oggetto è creato mediante l'operatore new

Se provate ad utilizzare un riferimento con valore null, otterrete un errore durante l'esecuzione del programma

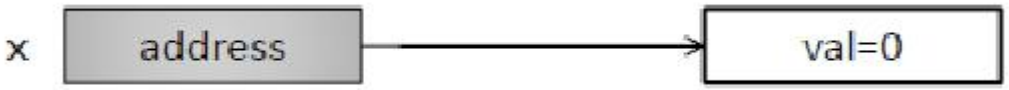
Esempio:

```
public static void main(String args[]){
    SimpleCounter x;
    x = new SimpleCounter();
    x.reset();
    x.inc();
    System.out.println("X = "+x.getValue());
}
```

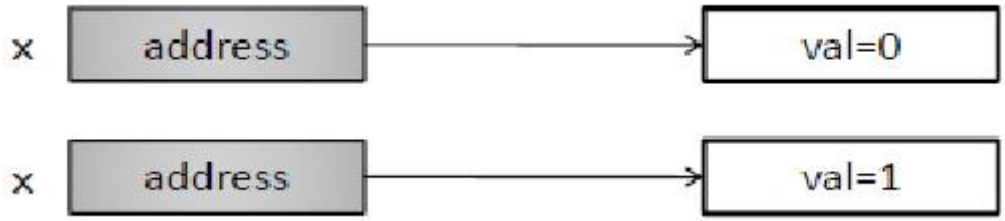
Riferimenti

Dichiaro un riferimento (x vale null) **Counter x;**

Creo un nuovo oggetto con new e inizializzo x **x = new Counter();**



Invoco reset() e inc() sull'oggetto attualmente puntato da x



Operatore di uguaglianza

```
SimpleCounter c1 = new
   SimpleCounter();
SimpleCounter c2 = c1;
if(c1==c2)
   System.out.println("Si");
else
   System.out.println("No");
```

Questo stampa

Si

```
SimpleCounter c1 = new
   SimpleCounter();
SimpleCounter c2 = new
   SimpleCounter();
if(c1==c2)
   System.out.println("Si");
else
   System.out.println("No");
```

Questo stampa

No

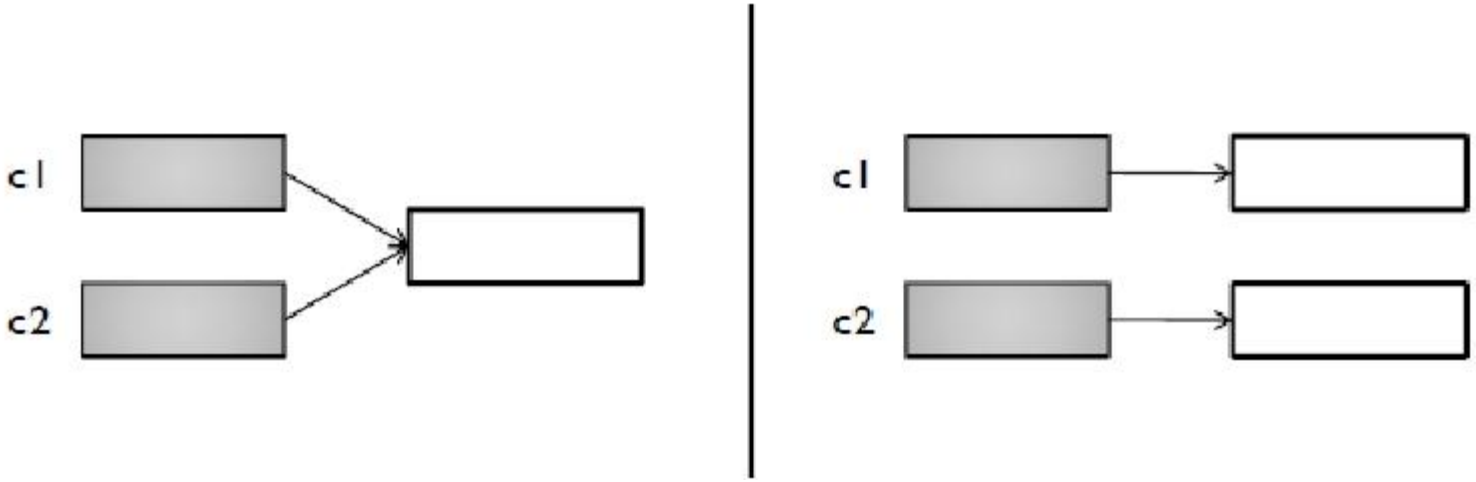
Operatore di uguaglianza

Nel caso di oggetti, l'operatore di uguaglianza verifica l'uguaglianza dei riferimenti.

Quindi due riferimenti sono uguali solo se essi puntano allo stesso oggetto (stesso indirizzo di memoria).

Nel primo caso, i riferimenti (indirizzi) sono uguali per via dell'assegnamento

Nel secondo caso, i riferimenti sono diversi: il new alloca due istanze diverse di contatore!



Attenzione ai riferimenti

Riprendiamo **il primo esempio** ed estendiamolo in:

```
SimpleCounter c1 = new SimpleCounter();
SimpleCounter c2 = c1;
c2.inc();
System.out.println("c1 vale: " + c1.getValue());
```

L'oggetto é condiviso: entrambi i riferimenti puntano allo stesso oggetto

Questo significa che viene stampato --> c1 vale: 1

I riferimenti c1 e c2 permettono di accedere/modificare la stessa istanza della classe SimpleCounter

Uguaglianza tra dati primitivi

Attenzione: per i tipi primitivi, le cose sono diverse. La variabile contiene il valore e non l'indirizzo di memoria a cui è allocato … Cosa restituisce quindi questo listato?

```
int c1 = 5;
int c2 = 5;
if(c1==c2)
        System.out.println("Si");
else
System.out.println("No");
```

In questo caso viene ovviamente stampato Si. Questo perché si tratta di un confronto fra variabili intere (quindi di tipo primitivo) Questi non sono oggetti!

L'operatore d'uguaglianza confronta il valore.

Uguaglianza tra oggetti

Ma allora come possiamo esprimere la condizione di uguaglianza di due oggetti?

È semplice … definiamo un metodo apposito che testa l'uguaglianza tra l'oggetto su cui viene invocato e l'oggetto passato come parametro

```
public boolean equals(SimpleCounter c)
```

Ad esempio

```
public class SimpleCounter {
        private int val;
        …
        public boolean equals(SimpleCounter c) {
                return (val == c.val);
        }
}
```

Uguaglianza tra oggetti

Considerando ora il seguente codice:

```
SimpleCounter c1 = new SimpleCounter();
SimpleCounter c2 = new SimpleCounter();
if(c1.equals(c2))
        System.out.println("Si");
else
        System.out.println("No");
```

Il risultato ora sarà Si

All'interno del metodo equals, possiamo definire noi il criterio di uguaglianza fra due oggetti. Ad esempio, potevamo dire che due contatori erano uguali a prescindere dal proprio valore … basta che il metodo ritorni sempre true

I costruttori

Molti errori di programmazione sono causati dalla mancata inizializzazione delle variabili

Nella programmazione ad oggetti, **il costruttore é il metodo che automatizza l'inizializzazione dell'oggetto**

- Non viene mai chiamato esplicitamente dall'utente
- Viene invocato implicitamente dal sistema ogni volta che si crea un nuovo oggetto mediante l'operatore new

Un costruttore

- deve avere lo stesso nome della classe
- non ha tipo di ritorno (nemmeno void). D'altronde, il suo compito é solo quello di inizializzare l'oggetto

Ogni classe può avere da zero ad infiniti costruttori (purché essi siano differenziati per lista dei parametri)

I costruttori

```
public class SimpleCounter {
        private int val;
        public SimpleCounter(){
                val = 0;
        }
        public SimpleCounter(int value) {
                val = value;
        }
}
```

Potete ora creare un oggetto scrivendo “new SimpleCounter()”. In questo caso, il contatore ha valore iniziale 0 (viene invocate il primo costruttore)

Altrimenti, se scrivete “new SimpleCounter(5)”, create un contatore con valore iniziale 5 (viene invocate il secondo costruttore)

I costruttori

Il costruttore senza parametri é detto di default

Anche se non dichiarato, la JVM lo fornisce automaticamente.

E' il costruttore che viene usato quando non si passano i parametri

Il costruttore di default fornito dalla JVM fa pochissimo: in pratica, inizializza tutti i singoli campi dell'oggetto secondo particolari convenzioni (ad esempio 0 per i numeri, false per i boolean e null per i campi riferimento)

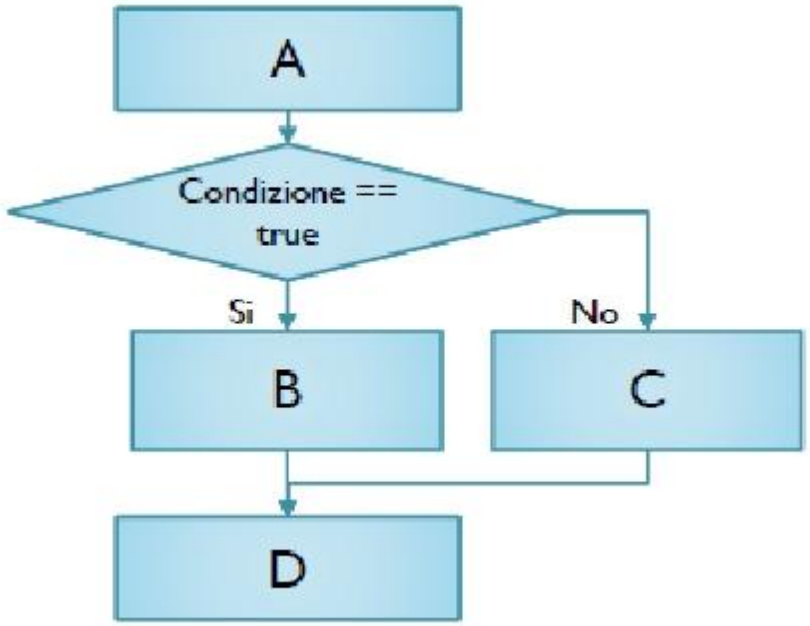
Controllo di flusso

L'esecuzione delle istruzioni avviene in maniera sequenziale, per modificare il flusso di esecuzione bisogna utilizzare apposite istruzioni

Il **costrutto if** consente di eseguire del codice a seconda che la condizione di test fornita è vero o falsa

Il blocco else può essere omesso: in questo caso, se la condizione è falsa, si esegue direttamente il codice dopo l'if (blocco D)

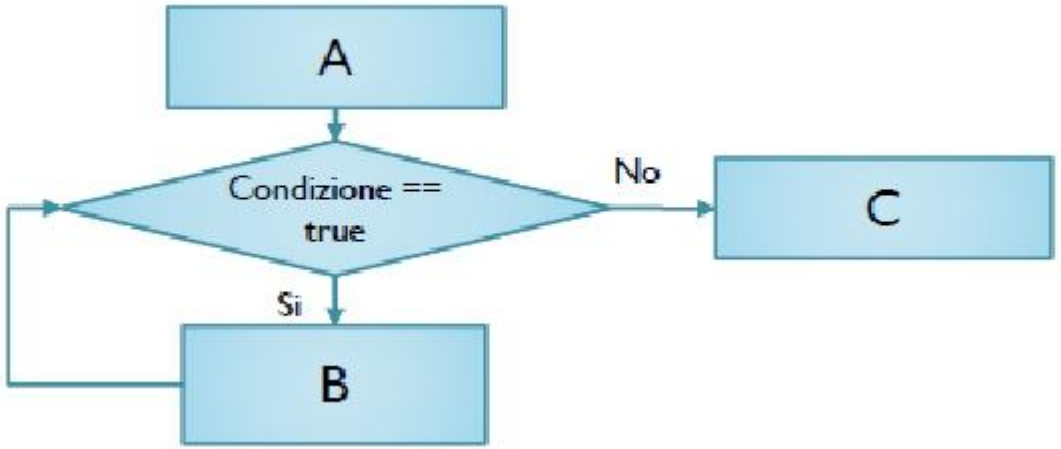
```
//codice   A

if(condizione){
    //codice   B
}
else{
    //codice   C
}

//codice   D
```



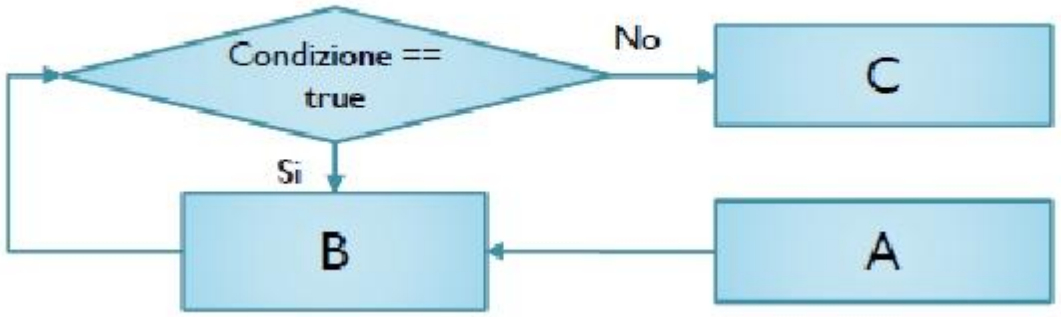
Controllo di flusso

Le istruzioni **while** e **do...while** consentono di eseguire ripetutamente un blocco di codice fino a che una condizione è vera
La formulazione con while esegue il blocco da 0 a più volte, mentre il do…while da 1 a più volte

```
//codice   A
while(condizione){
    //codice   B
}
//codice   C
```




```
//codice   A
do{
    //codice   B
}while(condizione);
//codice   C
```



Controllo di flusso

Java offre l'istruzione **for …** del tutto equivalente ad un'istruzione while: consente di specificare contemporaneamente inizializzazione della variabile condizione, test di terminazione e istruzione di aggiornamento della variabile condizione

for(*inizializzazione*; *condizione d'uscita*; *aggiornamento*){
//codice
}

Ad esempio, i seguenti blocchi di codice sono del tutto equivalenti

```
for(int i=0; i<10; i++){
    //codice
}
```

```
int i=0;
while(i<10){
    //codice
    i++;
}
```

Esercizio 1

Create un **tipo di dato Counter** che abbia:

un valore attuale

un valore massimo di conteggio

uno stato interno che indica se si è verificato un errore. Il valore massimo è selezionabile dall'utente alla costruzione. Se si tenta di superarlo, viene modificato lo stato del contatore per memorizzare l'avvenuta condizione d'errore e le successive operazioni di modifica non devono avere effetto

Il contatore deve offrire **i seguenti metodi:** void inc(),void reset(),int getValue(),boolean isError()

Infine, scrivete un'applicazione Java che:

Crea e inizializza un nuovo Counter con valore massimo n
Incrementa il contatore per n+1 volte e visualizza, ogni volta, il valore attuale e lo stato interno (errato/corretto)
Terminare

Soluzione Esercizio 1

```
public class ContatoreEsteso {

private int valore;
private int valore_massimo;
private boolean errore;

//Costruttore
public ContatoreEsteso(int vm){
//Limite massimo del contatore
valore_massimo=vm;
//Valore iniziale
valore=0;
//E' possibile incrementare il contatore
errore=false;
}
```

```
 //Metodo per incrementare il valore
void inc(){
    //Se il contatore è in stato di errore, esco senza
     cambiare il valore
if (errore) return;
// Se tento di incrementare oltre il limite, setto errore true
if (valore==valore_massimo)
{errore=true;}
//Incremento il valore del contatore di 1
else valore=valore+1;
      }
//Metodo per resettare il valore
    void reset(){
//Porto a 0 il valore del contatore
valore=0;
//Porto a 0 il valore del contatore
errore=false;
}
```

```
//Metodo per ottenere il valore del contatore
int getValue(){
return(valore);
}

   //Metodo per ottenere informazioni sullo stato interno
boolean isError(){
 return errore;
}

//Metodo main
public static void main(String[] args) {
        ContatoreEsteso c= new ContatoreEsteso(5);
        for(int   i=0;   i<6;   i++){
        c.inc();
        System.out.println("Valore del contatore: " + c.getValue());
           System.out.println("Stato di errore: " + c.isError());
        }
}
}
```

Esercizio 2: classe Rettangolo

Creare un **tipo di dato Rettangolo** tale che un oggetto sia definito da:

- Un intero per la base
- Un intero per l'altezza
- Un intero per la coordinata di ascissa x
- Un intero per la coordinata di ordinata y

- Due costruttori: uno che crea rettangoli con dimensioni fissate, il secondo che permette all'utente di inizializzare i valori.

- Metodi: li dividiamo in più categorie
  - Metodi contenenti la dicitura get che restituiscono il valore della dimensione richiesta (es. getAltezza…) oppure effettuano un'operazione specifica (ad es. per il calcolo dell'area o del perimetro)
  - Metodi contenenti la dicitura set permettono di assegnare un nuovo valore alla variabile d'istanza (ad es. setAltezza)

Creare poi un main che crei un oggetto rettangolo e ne visualizzi perimetro e area.