itconsulting
Consulenza & Formazione

Java Continue

L'istruzione continue interrompe un'iterazione (nel ciclo), se si verifica una condizione specificata, continua con l'iterazione successiva nel ciclo.

Questo esempio salta il valore di 4:

```
1. public class Main {
2.   public static void main(String[] args) {
3.     for (int i = 0; i < 10; i++) {
4.       if (i == 4) {
5.         continue;
6.       }
7.       System.out.println(i);
8.     } }}
```

itconsulting
Consulenza & Formazione

Java Continue

Trovate la differenza.

```
1. public class Main {
2.   public static void main(String[] args) {
3.     int i = 0;
4.     while (i < 10) {
5.       if (i == 4) {
6.         i++;
7.         continue;
8.       }
9.       System.out.println(i);
10.      i++;
11.    }}}
```

itconsulting
Consulenza & Formazione

Java Arrays

Gli array vengono utilizzati per memorizzare più valori in una singola variabile, invece di dichiarare variabili separate per ciascun valore.

Per dichiarare un array, definisci il tipo di variabile con é [ ] :

```
String[] cars;
```

Abbiamo ora dichiarato una variabile che contiene un array di stringhe. Per inserire valori in esso, puoi inserire i valori in un elenco separato da virgole, all'interno di parentesi graffe:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

Per creare un array di numeri interi, puoi scrivere:

```
int[] myNum = {10, 20, 30, 40};
```

LA PAUSA FINISCE E 20

itconsulting
Consulenza & Formazione

Accedi agli elementi di un array

È possibile accedere a un elemento dell'array facendo riferimento al numero di indice.

Questa affermazione accede al valore del primo elemento nelle automobili:

```
1. public class Main {
2.   public static void main(String[] args) {
3.     String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
4.     System.out.println(cars[0]);
5.   }
6.}
```

Nota: gli indici dell'array iniziano con 0: [0] è il primo elemento. [1] è il secondo elemento, ecc.

itconsulting
Consulenza & Formazione

Cambia un elemento dell'array

Per modificare il valore di un elemento specifico, fare riferimento al numero di indice:

```
1. public class Main {
2.   public static void main(String[] args) {
3.     String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
4.     cars[0] = "Fiat";
5.     System.out.println(cars[0]);
6.   }
7.}
8.
```

itconsulting
Consulenza & Formazione

Lunghezza matrice

Per scoprire quanti elementi ha un array, usa la proprietà length :

```
1. public class Main {
2.   public static void main(String[] args) {
3.     String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
4.     System.out.println(cars.length);
5.   }
6.}
```

itconsulting
Consulenza & Formazione

Loop attraverso una matrice

È possibile scorrere gli elementi dell'array con il forciclo e utilizzare la length proprietà per specificare quante volte il ciclo deve essere eseguito

L'esempio seguente emette tutti gli elementi nell'array cars :

```
1. public class Main {
2.   public static void main(String[] args) {
3.     String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
4.     for (int i = 0; i < cars.length; i++) {
5.       System.out.println(cars[i]);
6.     }
7.   }
8. }
```

itconsulting
Consulenza & Formazione

Multidimensional Arrays

Un array multidimensionale è un array di array.

Gli array multidimensionali sono utili quando si desidera archiviare i dati sotto forma di tabella, come una tabella con righe e colonne.

Per creare un array bidimensionale, aggiungi ciascun array all'interno del proprio set di parentesi graffe :

```
int[]myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

myNumbers è ora un array con due array come elementi.

itconsulting
Consulenza & Formazione

Elementi di accesso

Per accedere agli elementi dell'array myNumbers , specificare due indici: uno per l'array e uno per l'elemento all'interno dell'array. Questo esempio accede al terzo elemento (2) nel secondo array (1) di myNumbers:

```
1. public class Main {
2.   public static void main(String[] args) {
3.     int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
4.     System.out.println(myNumbers[1][2]);
5.   }
6. }
```

itconsulting
Consulenza & Formazione

Scorrere un array multidimensionale

```
1. public class Main {
2.    public static void main(String[] args) {
3.      int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
4.      for (int i = 0; i < myNumbers.length; ++i) {
5.        for(int j = 0; j < myNumbers[i].length; ++j) {
6.          System.out.println(myNumbers[i][j]);
7.        }
8.      }
9.    }
10. }
```

itconsulting
Consulenza & Formazione

ArrayList

Possiamo usare ArrayList come struttura intermedia e aggiungere gli elementi nell'ArrayList usando il metodo add().

ArrayList è una struttura dati che ci consente di aggiungere dinamicamente elementi.

Tuttavia, possiamo convertire l'ArrayList nell'array utilizzando il metodo toArray().

Quindi questo processo prevede i seguenti passaggi.

itconsulting
Consulenza & Formazione

ArrayList

Converti Array in ArrayList usando il metodo asList().Aggiungere elementi all'elenco di array utilizzando il metodo add().Converti nuovamente ArrayList nell'array utilizzando il metodo toArray().

```
1. import  java.util.ArrayList;
2. import  java.util.Arrays;
3. class pubblic  JavaAddElementUsingList {
4. public static void  main(String[] args) {
5. Integer arr[] = { 1 , 2 , 3 , 4 , 5 , 6 };
6. System.out.println( "Array:" +Arrays.toString(arr));
7. ArrayList<Integer> arrayList =  new  ArrayList<Integer>(Arrays.asList(arr));
8. arrayList .add( 7 );
9. arr = arrayList .toArray(arr);
10. System.out.println( "Array dopo aver aggiunto l'elemento: " +Arrays.toString(arr));
11. } }
```

itconsulting
Consulenza & Formazione

Esercizio

Creare un array numeri e un array stringhe.

Far si che tramite un menu si possa scegliere prima quali visualizzare e poi a quale aggiungere un elemento per poi tornare al menu iniziale

itconsulting
Consulenza & Formazione

ArrayList

```
1. import java.util.ArrayList;
2.
3. public class Main {
4.   public static void main(String[] args) {
5.     ArrayList<String> cars = new ArrayList<String>();
6.     cars.add("Volvo");
7.     cars.add("BMW");
8.     cars.add("Ford");
9.     cars.add("Mazda");
10.    System.out.println(cars);
11.  }
12.}
```

itconsulting
Consulenza & Formazione

ArrayList

```
1. import java.util.ArrayList;
2.
3. public class Main {
4.   public static void main(String[] args) {
5.     ArrayList<String> cars = new ArrayList<String>();
6.     cars.add("Volvo");
7.     cars.add("BMW");
8.     cars.add("Ford");
9.     cars.add("Mazda");
10.    cars.set(0, "Opel");
11.    System.out.println(cars);
12.    System.out.println(cars.get(0));
13.  }
14.}
```

itconsulting
Consulenza & Formazione

ArrayList

```
1. import java.util.ArrayList;
2.
3. public class Main {
4.   public static void main(String[] args) {
5.     ArrayList<String> cars = new ArrayList<String>();
6.     cars.add("Volvo");
7.     cars.add("BMW");
8.     cars.add("Ford");
9.     cars.add("Mazda");
10.    System.out.println(cars.size());
11.    cars.remove(0);
12.    cars.clear();
13.  } }
```

itconsulting
Consulenza & Formazione

Metodi in Java

Un metodo è un blocco di codice che viene eseguito solo quando viene chiamato.

È possibile passare dati, noti come parametri, in un metodo.

I metodi vengono utilizzati per eseguire determinate azioni e sono noti anche come funzioni .

Perché usare i metodi? Per riutilizzare il codice: definisci il codice una volta e usalo molte volte.

itconsulting
Consulenza & Formazione

Metodi in Java

Un metodo deve essere dichiarato all'interno di una classe. È definito con il nome del metodo, seguito da parentesi () .
Java fornisce alcuni metodi predefiniti, come System.out.println(), ma puoi anche creare i tuoi metodi per eseguire determinate azioni:

Crea un metodo all'interno di Main:

```
1. public class Main {
2.   static void myMethod() {
3.     // code to be executed
4.   }
5.}
```

- **myMethod() è il nome del metodo**
- **static -significa che il metodo appartiene alla classe Main e non un oggetto della classe Main.**
- **void -significa che questo metodo non ha un valore di ritorno. Imparerai di più sui valori restituiti più avanti in questo capitolo**

itconsulting
Consulenza & Formazione

Metodi in Java

Un metodo deve essere dichiarato all'interno di una classe. È definito con il nome del metodo, seguito da parentesi () .
Java fornisce alcuni metodi predefiniti, come System.out.println(), ma puoi anche creare i tuoi metodi per eseguire determinate azioni:

Crea un metodo all'interno di Main:

```
1. public class Main {
2.   static void myMethod() {
3.     // code to be executed
4.   }
5.}
```

- **myMethod() è il nome del metodo**
- **static -significa che il metodo appartiene alla classe Main e non un oggetto della classe Main.**
- **void   -significa che questo metodo non ha un valore di ritorno. Imparerai di più sui valori restituiti più avanti in questo capitolo**

Metodi in Java

Parametri e Argomenti

Le informazioni possono essere passate ai metodi come parametro. I parametri agiscono come variabili all'interno del metodo.

I parametri vengono specificati dopo il nome del metodo, all'interno delle parentesi. Puoi aggiungere tutti i parametri che vuoi, basta separarli con una virgola.

itconsulting
Consulenza & Formazione

Metodi in Java

l'esempio seguente ha un metodo che accetta un fnameString chiamato come parametro. Quando viene chiamato il metodo, passiamo un nome, che viene utilizzato all'interno del metodo per stampare il nome completo:

```
1. public class Main {
2.   static void myMethod(String fname) {
3.     System.out.println(fname + " Refsnes");
4.   }
5.   public static void main(String[] args) {
6.     myMethod("Liam");
7.     myMethod("Jenny");
8.     myMethod("Anja");
9.   }}
```

Quando un parametro viene passato al metodo, viene chiamato argomento . Quindi, dall'esempio precedente: fnameis a parameter , while Liam, Jennye Anjaare arguments .

itconsulting
Consulenza & Formazione

Valori di ritorno - Return

La voidparola chiave, utilizzata negli esempi precedenti, indica che il metodo non deve restituire un valore.

Se vuoi che il metodo restituisca un valore, puoi usare un tipo di dati primitivo (come int, char, ecc.) invece di void, e usare la return parola chiave all'interno del metodo:

```
1. public class Main {
2.   static int myMethod(int x) {
3.     return 5 + x;
4.   }
5.   public static void main(String[] args) {
6.     System.out.println(myMethod(3));
7.   }
8. }
```

itconsulting
Consulenza & Formazione

Valori di ritorno - Return

Questo esempio restituisce la somma dei due parametri di un metodo :

```
1. public class Main {
2.   static int myMethod(int x, int y) {
3.     return x + y;
4.   }
5.
6.   public static void main(String[] args) {
7.     System.out.println(myMethod(5, 3));
8.   }
9. }
10.
```

itconsulting
Consulenza & Formazione

Altre caratteristiche

Tra tutti i qualificatori utilizzabili nella dichiarazione di un metodo quello che più di ogni altro ne modifica il funzionamento (nel senso che chiariremo tra poco) è static.
Per comprendere l'uso della keyword static in Java bisogna ricordare innanzi tutto che, come detto, ogni metodo appartiene ad una classe ed una classe è, in qualche modo, un "pacchetto" di dati e metodi.
Sappiamo che da una classe possiamo ottenere molteplici istanze e per ciascuna istanza si hanno variabili dai nomi identici ma dai valori distinti (forse "che puntano a locazioni di memoria diverse" sarebbe una definizione più chiara).
Se poi vogliamo che una variabile sia la medesima per tutte le istanze di una classe sappiamo che la dobbiamo invece definire come static.
Per i metodi avviene sostanzialmente la medesima cosa: possiamo pensare che dei metodi definiti in una classe ne esista normalmente (cioè se non si specifica static) una "copia" per ogni istanza della classe, mentre dei metodi statici ne esista una sola copia associata alla classe stessa. Per scendere più in dettaglio:

- **i metodi non statici sono associati ad ogni singola istanza di una classe e perciò il loro contesto di esecuzione (quindi l'insieme delle variabili cui possono accedere) è relativo all'istanza stessa: possono accedere e modificare le variabili dell'istanza e modificarne lo stato;**
- **in contapposizione i metodi statici non sono associati ad una istanza ma solo ad una classe. Quindi non potranno interagire con le variabili di istanza, ma solamente con quelle statiche.**

Questa distinzione tra metodi statici e metodi di istanza si riflette anche in una diversa sintassi che si deve utilizzare per eseguire i 2 tipi di metodi:

STATICO	NON STATICO
NomeClasse.nomeMetodo(...)	nomeIstanza.nomeMetodo(...)

itconsulting
Consulenza & Formazione

Altre caratteristiche

Come abbiamo detto parlando dei metodi, tra le loro caratteristiche note di Java possiamo citare quella dell'overloading , che ci consente di "sovraccaricare" un metodo di una classe con diverse varianti, in base ai parametri passati come riferimento. Dal punto di vista dell'utilizzo della classe ciò ci consente di definire in maniera dinamica il metodo che meglio si adatta alla nostra esigenza.

```
1. /* Esempio di una classe con un costruttore*/
2. public class Prodotto{
3.  private int id;
4.  public Prodotto(int id, String desc)
5.  {
6.   // ...}
7.  public Prodotto(int id, String desc1, String desc2)
8.  {
9.   // ... }
10. public Prodotto(int id, String desc1, String desc2, String desc3)
11. {
12.  // ...}}
```