

# Indice modulo 2

---

```
public class Main {

    int x = 3; // dichiarazione
        x = 5; // assegnamento
        // Assegnamento == modifica, non inizializzazione
        // di una variabile

    public Main(){
        System.out.println("ciao");
    }

    public static int f(int n){
        return n + 1;
    } // tecnicamente è un metodo, non una funzione

    public int g(int n){
        return n + 1;
    }

    public static void main(String[] args) {
        int x = f(10);
        // garbage collector cancella poi l'oggetto
        int y = new Main().g(13);
        // senza cancellazione
        Main o = new Main();
        int y = o.g(13);
        // esplode
    }
}
```

```

        // il nome di una variabile dice dove, non cosa
        Main o;
        int y = o.g(7);
    }
}

```

```

package it.unive.dais.po2.misc;

public class Main {
    class Zoo {
        public static class Animal {
            protected int weight;

            public Animal(int w) { this.weight = w; }

            public void eat(Animal a) {
                this.weight += a.weight;
            }
        }
    }

    static class Dog extends Animal { // sottoclasse
        private boolean pedigree;

        public Dog(boolean pedigree, int w){
            this.pedigree = pedigree;
            super(w);
        }

        public void bark() {
            System.out.println("bau!");
        }
    }
}
@Override

```

```

        //Subsumption
        // viene chiamata questa funzione da pluto,
        // anche se è un animale, non un cane
        public void eat(Animal a){
            this.weight += a.weight * 2;
        }
    }

    public static void main(String[] args){
        Dog fido = new Dog(false, 30);
        Dog gigio = fido;
        Animal pluto = new Dog(true, 40); // polimorfismo
        gigio.bark();
        // non funziona
        // pluto è animal, anche se possiede il metodo
        pluto.bark();
        // Dynamic dispatching, chiama override
        // non hai perdita di informazioni a runtime
        pluto.eat(fido);
        gigio.eat(gigio);
    }
}

```

Implementare == Estendere

## Documentazione di Java

Link per la documentazione: <https://docs.oracle.com/en/java/javase/17/docs/api/>

SottoInterfacce disponibili dalla libreria:

**All Known Subinterfaces:**

BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Collection<E>, Deque<E>, DirectoryStream<T>, EventSet, List<E>, NavigableSet<E>, NodeSetData<T>, Path, Queue<E>, SecureDirectoryStream<T>, Set<E>, SortedSet<E>, TransferQueue<E>, XPathNodes

Sotto classi che implementano l'interfaccia

----- / -----

All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BatchUpdateException, BeanContextServicesSupport, BeanContextSupport, ConcurrentHashMap, KeySetView, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DataTruncation, DelayQueue, DocTreePath, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, RowSetWarning, SerialException, ServiceLoader, SQLClientInfoException, SQLDataException, SQLException, SQLFeatureNotSupportedException, SQLIntegrityConstraintViolationException, SQLInvalidAuthorizationSpecException, SQLNonTransientConnectionException, SQLNonTransientException, SQLRecoverableException, SQLSyntaxErrorException, SQLTimeoutException, SQLTransactionRollbackException, SQLTransientConnectionException, SQLTransientException, SQLWarning, Stack, SyncFactoryException, SynchronousQueue, SyncProviderException, TreePath, TreeSet, Vector

Funzioni della interfaccia/classe di riferimento:

Method Summary

All Methods	Instance Methods	Abstract Methods	Default Methods
Modifier and Type	Method	Description	
default void	<code>forEach(Consumer&lt;? super T&gt; action)</code>	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.	
<code>Iterator&lt;T&gt;</code>	<code>iterator()</code>	Returns an iterator over elements of type T.	
default <code>Spliterator&lt;T&gt;</code>	<code>spliterator()</code>	Creates a <code>Spliterator</code> over the elements described by this Iterable.	

Descrizione funzioni:

## iterator

```
Iterator<T> iterator()
```

Returns an iterator over elements of type T.

**Returns:**  
an Iterator.

## forEach

```
default void forEach(Consumer<? super T> action)
```

Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception. Actions are performed in the order of iteration, if that order is specified. Exceptions thrown by the action are relayed to the caller.

The behavior of this method is unspecified if the action performs side-effects that modify the underlying source of elements, unless an overriding class has specified a concurrent modification policy.

### Implementation Requirements:

The default implementation behaves as if:

```
for (T t : this)
    action.accept(t);
```

### Parameters:

action - The action to be performed for each element

### Throws:

`NullPointerException` - if the specified action is null

### Since:

1.8

A cosa serve la keyword default?

- Implementazione di default, disponibile da java 11

```
public class Misc1 {
    // Interfaccia == Classe astratta
    public interface I { // Interfaccia
        void a();
        void b();
        default void c(){
            a();
            b();
        }
    }
}
```

```
public static abstract class J { // Classe astratta
    public abstract void a();
    public abstract void b();
    public void c(){
        a();
        b();
    }
}

public static class C implements I {
    @Override
    public void a() {

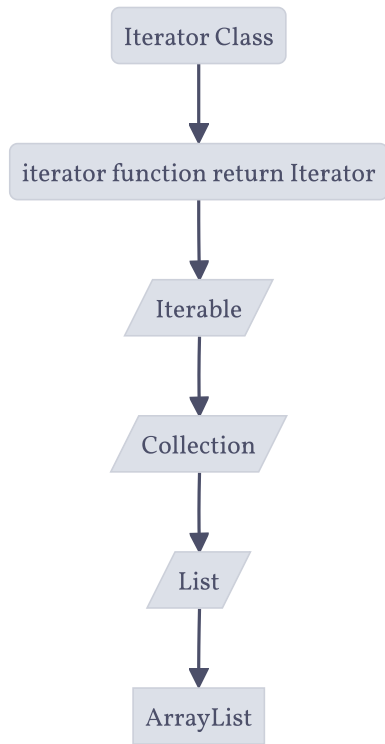
    }

    @Override
    public void b() {

    }
}

public static void main(String[] args){
    I o = new C();
}
}
```

Interfaccia valida? Si



```
import java.util.ArrayList;

public class Misc2 {
    public static void main(String[] args){
        // No subsumption
        // costruttore vuoto/di default
        // vuoto se implementato dal programmatore
        // senza parametri in input
        ArrayList<Integer> l = new ArrayList<Integer>();
        // add ereditato da Collection
        l.add(21);
        l.add(2);
        l.add(456);
    }
}
```

```

List<Integer> g = new ArrayList<Integer>();
// con Collection non funziona
// la classe non ha set(), List e ArrayList
// invece si
g.add(21);
g.add(2);
g.add(456);
g.set(2,987);

// ciclo for classico
for(int i=0; i < g.size(); i++){
    int n = g.get(i);
    System.out.println(n);
}

// ciclo con iteratore
Iterator<Integer> it = g.iterator();
while (it.hasNext()){
    int n = it.next();
    System.out.println(n);
}

// con subsumption
Collection<Integer> m = new ArrayList<Integer>();
m.add(21);
m.add(2);
m.add(456);
}
}

```

- type parameter
  - void f(int n) { ... }
- type argument



- `f(7)`

```
// Iterable.java

package tinyjdk;

// ogni file contiene una sola classe
// al massimo sono nested una dentro l'altra
public interface Iterable<T> { // <T> = parameter
    Iterator<T> iterator();
}
```

```
// Iterator.java

package tinyjdk;

public interface Iterator<T> {
    boolean hasNext();
    T next(); // ritorna un oggetto generico
}
```

- Ipotezzando di non voler usare i *Generics* all'interno di `Collection.java`, dovremmo programmare usando la classe *Object*, ed essa permette di poter aggiungere tramite la `add` qualunque cosa che prende in input, elevando tutto ciò che riceve appunto, ad *Object*, ma il problema si pone quando bisogna gettare, facendo un subcast della classe e rischiando errori dovuti ad esso. Per questo, si usano i *Generics*, che bloccano ad un certo type

```
// Collection.java

package tinyjdk;

// ogni file contiene una sola classe
// al massimo sono nested una dentro l'altra
```

```
public interface Collection<T> extends Iterable<T>{

    void add(T x);

    default void addAll(Collection<T> c){
        Iterator<T> it = c.iterator();
        while (it.hasNext()){
            add(it.next())
        }
    }

    void clear();

    boolean contains(T x);

    boolean isEmpty();

    void remove(T x);

    int size();

}
```

//List.java

```
package tinyjdk;

public interface List<T> extends Collection<T> {

    T get(int i);

    T set(int i, T x);

    void add(int i, T x);

}
```

```
    T remove(int i);  
}
```

```
// ArrayList.java  
  
package tinyjdk;  
  
public class ArrayList<T> implements List<T> {  
    private Object[] a;  
    private int sz;  
  
    public ArrayList(){  
        this.a = new Object[10];  
        sz = 0;  
    }  
  
    public ArrayList(int i){  
  
    }  
  
    @Override  
    public void add(T x){  
        if(sz >= a.length){  
            Object[] old = a;  
            a = new Object[a.length * 2];  
            for(int i=0; i < old.length; i++){  
                a[i] = old[i];  
            }  
        }  
        a[sz++] = x;  
    }  
  
    @Override
```

```
public void clear(){  
    sz = 0; // non disalloco, tanto non mi interessa  
}
```

```
@Override  
public T get(int i){  
    return (T) a[i];  
}
```

```
}
```