Metodi di uso frequente della classe di libreria `java.lang.String` (classe immutabile, nel senso che i suoi oggetti non sono più modificabili dopo la creazione)

- `String(String other)` (costruttore di copia: crea un clone)
- `char charAt(int index)`
- `int compareTo(String other)` (ritorna negativo, zero, oppure positivo)
- `int compareToIgnoreCase(String other)` (ritorna negativo, zero, oppure positivo)
- `String concat(String other)` (implicitamente usato per la concatenazione con `+`)
- `boolean endsWith(String end)`
- `boolean equals(Object other)`
- `boolean equalsIgnoreCase(String other)`
- `static String format(String format, Object... args)`
- `int indexOf(int character)`
- `int indexOf(String what)`
- `boolean isEmpty()`
- `int length()`
- `boolean startsWith(String what)`
- `String substring(int start)` (da `start` incluso)
- `String substring(int start, int end)` (da `start` incluso ad `end` escluso)
- `String toLowerCase()`
- `String toUpperCase()`
- `String trim()`
- `static String valueOf(int i)` (esegue una conversione esplicita di tipo; esiste per tutti i tipi primitivi, non solo per `int`; implicitamente usato per la concatenazione con `+`)

Metodi di uso frequente della classe di libreria `java.lang.System`

- `err` (costante della classe che fa riferimento allo standard error)
- `in` (costante della classe che fa riferimento allo standard input)
- `out` (costante della classe che fa riferimento allo standard output)
- `static long currentTimeMillis()` (ritorna il numero di millisecondi passati dal mezzogiorno dell'1 gennaio 1970 UTC)

Metodi di uso frequente della classe di libreria `java.util.Scanner`

- `Scanner(source)` (costruttore, che crea uno `Scanner` legato alla sorgente indicata)
- `void close()` (chiude lo `Scanner`: dopo non può più essere usato)
- `double nextDouble()`
- `float nextFloat()`
- `int nextInt()`
- `String nextLine()`
- `long nextLong()`

Metodi di uso frequente della classe di libreria `java.util.Random`

- `boolean nextBoolean()`
- `double nextDouble()`
- `float nextFloat()`
- `int nextInt()`
- `int nextInt(int max)` (restituisce un numero casuale tra 0 e `max` escluso)
- `long nextLong()`

Metodi di uso frequente della classe di libreria `java.lang.Math`

- `static double E` (costante della classe)
- `static double PI` (costante della classe)
- `static int abs(int i)` (esiste anche per altri tipi numerici)
- `static double cos(double d)`
- `static double log(double d)` (in base e)
- `static double log10(double d)` (in base 10)
- `static int max(int a, int b)` (esiste anche per altri tipi numerici)
- `static int min(int a, int b)` (esiste anche per altri tipi numerici)
- `static double sin(double d)`
- `static double sqrt(double d)`
- `static double tan(double d)`
- `static double toDegrees(double radiants)`
- `static double toRadiants(double degrees)`

Metodi di uso frequente delle classi `E` definite tramite `enum`

- `static E[] values()` (ritorna l'array di tutti gli elementi dell'enumerazione)
- `static E valueOf(String name)` (ritorna l'elemento dell'enumerazione che ha il nome indicato)
- `int compareTo(E other)` (determina chi viene prima nell'enumerazione)
- `int ordinal()` (ritorna il numero d'ordine di un elemento dell'enumerazione)

Metodi di uso frequente della classe `java.util.Arrays`

- `static int binarySearch(int[] arr, int key)` (ritorna la posizione di `key` dentro `arr`, oppure un numero negativo se `arr` non contiene `key`. Assume che l'array `arr` sia ordinato. Questo metodo esiste anche per gli altri tipi primitivi numerici e per i tipi riferimento, nel qual caso chiama `compareTo()` per decidere l'ordine)
- `static boolean equals(int[] arr1, int[] arr2)` (controlla che `arr1` e `arr2` abbiano stessa lunghezza e contengano gli stessi elementi nello stesso ordine. Questo metodo esiste anche per gli altri tipi primitivi nonché per array di tipi riferimento, nel qual caso chiama `equals()` fra tutte le coppie di oggetti da confrontare)
- `static void fill(int[] arr, int val)` (assegna `val` a tutti gli elementi di `arr`. Questo metodo esiste anche per tutti gli altri tipi primitivi e per array di tipi riferimento)
- `static void sort(int[] arr)` (ordina `arr` in senso crescente, in tempo O(n log n). Questo metodo esiste anche per tutti gli altri tipi primitivi numerici e per i tipi riferimento, nel qual caso chiama `compareTo()` per decidere l'ordine)
- `static String toString(int[] arr)` (ritorna una stringa che riporta gli elementi di `arr`, nel loro ordine. Questo metodo esiste anche per gli altri tipi primitivi e per array di tipi riferimento, nel qual caso chiama `toString()` sugli elementi dell'array e concatena il risultato.

Metodi di uso frequente della classe `java.lang.Integer` (classe immutabile, nel senso che i suoi oggetti non sono più modificabili dopo la creazione)

- `static int MAX_VALUE` (costante che contiene il massimo `int` utilizzabile in Java)
- `static int MIN_VALUE` (costante che contiene il minimo `int` utilizzabile in Java)
- `Integer(int value)` (deprecato!)
- `Integer(String value) throws java.lang.NumberFormatException`
- `int intValue()` (restituisce il valore `int` corrispondente)
- `int compareTo(Integer other)` (infatti `Integer` implementa `Comparable<Integer>`)
- `static int parseInt(String s) throws java.lang.NumberFormatException` (traduce la stringa `s` in `int`)
- `static String toBinaryString(int i)` (ritorna la rappresentazione binaria di `i`)
- `static String toHexString(int i)` (ritorna la rappresentazione esadecimale di `i`)
- `static Integer valueOf(int i)` (ritorna `new Integer(i)` ma usa una cache per chiamate ripetute)

Esistono altre classi wrapper corrispondenti agli altri tipi primitivi, con costanti, costruttori e metodi simili a quanto riportato sopra: `java.lang.Short`, `java.lang.Long`, `java.lang.Float`, `java.lang.Double`, `java.lang.Byte` e `java.lang.Boolean`.

Metodi di uso frequente della classe `java.lang.Character` (classe immutabile, nel senso che i suoi oggetti non sono più modificabili dopo la creazione)

- `static char MAX_VALUE` (costante che contiene il massimo `char` utilizzabile in Java)
- `static char MIN_VALUE` (costante che contiene il minimo `char` utilizzabile in Java)
- `Character(char value)` (deprecato!)
- `char charValue()` (restituisce il valore `char` corrispondente)
- `int compareTo(Character other)` (infatti `Character` implementa `Comparable<Character>`)
- `static boolean isDigit(char c)`
- `static boolean isLetter(char c)`
- `static boolean isLetterOrDigit(char c)`
- `static boolean isLowerCase(char c)`
- `static boolean isUpperCase(char c)`
- `static boolean isWhitespace(char c)`
- `static char toLowerCase(char c)`
- `static char toUpperCase(char c)`
- `static Character valueOf(char c)` (ritorna `new Character(c)` ma usa una cache per chiamate ripetute)

Metodi di uso frequente dell'interfaccia `java.util.Collection<E>`

- `boolean add(E element)` (ritorna true se l'elemento viene aggiunto)
- `boolean addAll(Collection<E> other)` (ritorna true se almeno un elemento viene aggiunto)
- `boolean contains(Object element)`
- `boolean containsAll(Collection<?> other)`
- `boolean isEmpty()`
- `boolean remove(Object element)` (ritorna true se l'elemento viene rimosso)
- `boolean removeAll(Collection<?> other)` (ritorna true se almeno un elemento viene rimosso)
- `boolean retainAll(Collection<?> other)` (ritorna true se almeno un elemento viene rimosso)
- `int size()`

Metodi di uso frequente dell'interfaccia `java.util.List<E>` (oltre a quelli ereditati da `java.util.Collection<E>`)

- `boolean add(E element)` (aggiunge `element` in fondo alla lista, anche se la lista già lo conteneva; ritorna sempre true)
- `void add(int index, E element)` (piazza l'elemento alla posizione `index`, che deve essere fra 0 e `size()` inclusi, spostando di una posizione a destra l'elemento che c'era precedentemente e quelli alla sua destra)
- `E get(int index)` (ritorna l'elemento alla posizione `index`, che deve essere fra 0 incluso e `size()` escluso)
- `int indexOf(Object element)` (ritorna la prima posizione in cui occorre `element`; ritorna -1 se la lista non contiene `element`)
- `static <E> List<E> of(E... elements)` (factory method che costruisce una lista immutabile con dentro `elements`)
- `boolean remove(Object element)` (rimuove la prima occorrenza di `element`, se presente; ritorna true se l'elemento viene rimosso)
- `E remove(int index)` (rimuove e ritorna l'elemento alla posizione `index`, che deve essere fra 0 incluso e `size()` escluso; gli elementi alla sua destra vengono spostati di una posizione a sinistra)
- `E set(int index, E element)` (ritorna l'elemento alla posizione `index`, che deve essere fra 0 e `size()` inclusi, e lo sostituisce con `element`)

Metodi di uso frequente dell'interfaccia `java.util.Queue<E>` (oltre a quelli ereditati da `java.util.Collection<E>`)

- `E poll()` (rimuove e ritorna la testa della coda; ritorna `null` se la coda è vuota)
- `E remove() throws java.util.NoSuchElementException` (rimuove e ritorna la testa della coda, se non è vuota; altrimenti lancia un'eccezione)
- `E peek()` (ritorna la testa della coda, senza rimuoverla; ritorna `null` se la coda è vuota)
- `E element() throws java.util.NoSuchElementException` (ritorna la testa della coda, senza rimuoverla; se la code è vuota, lancia un'eccezione)
- `boolean offer(E element)` (aggiunge `element` in fondo alla coda, se c'è spazio. Ritorna `true` se e solo se l'elemento viene aggiunto)
- `boolean add(E element) throws java.lang.IllegalStateException` (aggiunge `element` in fondo alla coda, se c'è spazio, altrimenti lancia un'eccezione. Ritorna sempre `true`)

Metodi di uso frequente dell'interfaccia `java.util.Set<E>` (oltre a quelli ereditati da `java.util.Collection<E>`)

- `static <E> Set<E> of(E... elements)` (factory method)

Metodi di uso frequente della classe `java.util.LinkedList<E>`

- `LinkedList()`
- `LinkedList(Collection<? extends E> parent)` (crea una lista e la riempie con gli elementi di `parent`)

Metodi di uso frequente della classe `java.util.ArrayList<E>`

- `ArrayList()`
- `ArrayList(Collection<? extends E> parent)` (crea una lista e la riempie con gli elementi di `parent`)

Metodi di uso frequente della classe `java.util.PriorityQueue<E>` (oltre a quelli ereditati da `java.util.Queue<E>`). Si tratta di una coda unbounded: non ha un limite massimo di elementi ma si espande quando necessario.

- `PriorityQueue()`
- `PriorityQueue(Collection<? extends E> parent)` (crea una coda e la riempie con gli elementi di `parent`)

Metodi di uso frequente della classe `java.util.HashSet<E>` (oltre a quelli ereditati da `java.util.Set<E>`)

- `HashSet()`
- `HashSet(Collection<? extends E> parent)` (crea un insieme e lo riempie con gli elementi di `parent`)

Metodi di uso frequente dell'interfaccia `java.util.SortedSet<E>` (oltre a quelli ereditati o ridefiniti da `java.util.Set<E>`)

- `E first() throws java.util.NoSuchElementException` (ritorna, ma non rimuove, l'elemento più piccolo dell'insieme)
- `E last() throws java.util.NoSuchElementException` (ritorna, ma non rimuove, l'elemento più grande dell'insieme)

Metodi di uso frequente della classe `java.util.TreeSet<E>` (oltre a quelli ereditati o ridefiniti da `java.util.SortedSet<E>`)

- `TreeSet()`
- `TreeSet(Collection<? extends E> parent)` (crea un insieme ordinato e lo riempie con gli elementi di `parent`)

Metodi di uso frequente dell'interfaccia `java.lang.AutoCloseable`

- `void close() throws java.lang.Exception`

Metodi di uso frequente dell'interfaccia `java.io.Closeable`

- `void close() throws java.io.IOException`

Metodi di uso frequente dell'interfaccia `java.util.Map<K,V>`

- `boolean containsKey(Object key)`
- `boolean containsValue(Object value)`
- `V get(Object key)`
- `V getOrDefault(Object key, V defaultValue)`
- `boolean isEmpty()`
- `Set<K> keySet()`
- `V put(K key, V value)` (setta il valore per la chiave; rimpiazza il valore se la chiave era già presente; ritorna il vecchio valore, se già presente, altrimenti `null`)
- `V putIfAbsent(K key, V value)` (rimpiazza il valore ma solo se la chiave non era già presente; ritorna il vecchio valore, se già presente, altrimenti `null`)
- `V remove(Object key)` (ritorna il vecchio valore, se era presente; altrimenti `null`)
- `int size()`
- `Collection<V> values()`

Metodi di uso frequente della classe `java.util.HashMap<K,V>`

- `HashMap()`
- `HashMap(Map<? extends K,? extends V> parent)` (crea una mappa e la riempie con le coppie contenute in `parent`)

Metodi di uso frequente dell'interfaccia `java.util.SortedMap<K,V>` (oltre a tutti quelli di `Map`)

- `K firstKey()`
- `K lastKey()`

Metodi di uso frequente della classe `java.util.TreeMap<K,V>`

- `TreeMap()`
- `TreeMap(Map<? extends K,? extends V> parent)` (crea una mappa e la riempie con le coppie contenute in `parent`)

Metodi di uso frequente della classe astratta `java.io.Reader`

- `int read() throws java.io.IOException` (blocca l'esecuzione finché non c'è un carattere da leggere; a quel punto ritorna il codice Unicode del prossimo carattere letto; ritorna -1 se la sorgente di lettura è terminata)
- `int read(char[] buffer) throws java.io.IOException` (blocca l'esecuzione finché non arriva qualche carattere da leggere; a quel punto scrive i caratteri nel `buffer` e ritorna il numero di caratteri letti; ritorna -1 se la sorgente di lettura è terminata)

Metodi di uso frequente della classe `java.io.FileReader` (oltre quelli ereditati da `java.io.Reader`)

- `FileReader(String fileName) throws java.io.FileNotFoundException` (crea un lettore di file che legge i caratteri dal file di testo col nome indicato)

Metodi di uso frequente della classe `java.io.BufferedReader` (oltre quelli ereditati da `java.io.Reader`)

- `BufferedReader(Reader parent)` (crea una vista bufferizzata di `parent`)

Metodi di uso frequente della classe astratta `java.io.Writer`

- `void write(int c) throws java.io.IOException` (interpreta i 16 bit meno significativi di `c` come codice Unicode di un carattere e lo scrive nel file)
- `void write(char[] buffer) throws java.io.IOException` (scrive nel file i caratteri contenuti nell'array `buffer`)
- `void write(String s) throws java.io.IOException` (scrive nel file i caratteri della stringa `s`)

Metodi di uso frequente della classe `java.io.FileWriter` (oltre quelli ereditati da `java.io.Writer`)

- `FileWriter(String fileName) throws java.io.IOException` (crea uno scrittore di file che scrive i caratteri nel file di testo col nome indicato)

Metodi di uso frequente della classe `java.io.BufferedWriter` (oltre quelli ereditati da `java.io.Writer`)

- `BufferedWriter(Writer parent)` (crea una vista bufferizzata di `parent`)

Metodi di uso frequente della classe `java.io.PrintWriter` (oltre quelli ereditati da `java.io.Writer`)

- `PrintWriter(String fileName) throws java.io.FileNotFoundException` (crea uno scrittore di file che scrive i caratteri nel file di testo col nome indicato)
- `void print(int i)` (scrive i caratteri della rappresentazione decimale dell'intero `i` nel file. Questo metodo esiste anche per gli altri tipi primitivi)
- `void println(int i)` (scrive i caratteri della rappresentazione decimale dell'intero `i` nel file. Questo metodo esiste anche per gli altri tipi primitivi)
- `void print(String s)`
- `void println(String s)`
- `PrintWriter printf(String format, Object... args)` (scrive il formato nel file, in stile linguaggio C)

Metodi di uso frequente dell'interfaccia `java.lang.Iterable<T>`

- `Iterator<T> iterator()` (ritorna un iteratore, cioè un oggetto capace di restituire gli elementi dell'iterabile, uno alla volta)

Metodi di uso frequente dell'interfaccia `java.util.Iterator<T>`

- `boolean hasNext()`
- `T next()`

Metodi di uso frequente della classe `java.lang.Thread`

- `Thread()` (costruisce un thread che, quando verrà mandato in esecuzione, eseguirà il codice del suo metodo `run()`)
- `Thread(Runnable runnable)` (costruisce un thread che, quando verrà mandato in esecuzione, eseguirà il codice del metodo `run()` del `runnable`)
- `void start()` (manda in esecuzione il metodo `run()` del thread e ritorna immediatamente il controllo al chiamante; i due thread continueranno in parallelo)
- `void run()` (specifica il codice che viene eseguito in parallelo quando si chiama `start()`)
- `void join() throws java.lang.InterruptedException` (sospende il chiamante finché il thread destinatario non termina; se il chiamante viene interrotto mentre aspetta, viene lanciata una `InterruptedException`)
- `static void sleep(long milliseconds) throws InterruptedException` (sospende il chiamante per il numero di millisecondi indicato; se il chiamante viene interrotto mentre aspetta, viene lanciata una `InterruptedException`)
- `static Thread currentThread()` (ritorna il thread che sta eseguendo il codice chiamante)

Metodi di uso frequente dell'interfaccia `java.util.function.Consumer<T>`

- `void accept(T t)` (esegue del codice che usa `t`)

Metodi di uso frequente dell'interfaccia `java.util.function.Predicate<T>`

- `boolean test(T t)` (determina se `t` soddisfa il predicato)

Metodi di uso frequente dell'interfaccia `java.util.function.Supplier<T>`

- `T get()` (fornisce un oggetto di tipo `T`)

Metodi di uso frequente dell'interfaccia `java.util.function.Function<T,U>`

- `boolean U apply(T t)` (ritorna il valore della funzione applicata a `t`)