Oggetti dei tipi primitivi

String

Eccezioni
- Unchecked
- Checked
- Gestione
- Catch
- Finally
- Un esempio

Array e Liste
- Array
- ArrayList
- LinkedList
- ListIterator

Introduzione a Java (3)

Metodologie di Programmazione A.A. 2008-09

Stefano Guerrini

Corso di Laurea in Informatica
Dipartimento di Informatica.
Sapienza Università di Roma
guerrini@di.uniroma1.it

Febbraio - Marzo 2009

Oggetti dei tipi primitivi (1)

- In certi casi è comodo trattare valori dei tipi primitivi come oggetti.
- Per tutti i tipi primitivi il package java.lang fornisce una classe che racchiude in un oggetto un valore del corrispondente tipo primitivo.

int	**long**	**short**	**byte**	**double**	**float**	**char**	**boolean**
Integer	Long	Short	Byte	Double	Float	Character	Boolean

- Costruttori: Integer(7+2), Integer("13"), Boolean(**false**), Boolean("true"), Character('a')
- Il valore racchiuso dall'oggetto è mantenuto in un attributo privato accessibile mediante metodi getter.

```
Boolean o = new Boolean(true); . . . o.booleanValue() . . . ;
Integer o = new Integer("3"); . . . o.intValue() . . . ;
Double o = new Double(3.14); . . . o.doubleValue() . . . ;
```

- Tutte queste classi forniscono metodi per convertire un valore del corrispondente tipo base da/in una stringa, oltre a costanti e metodi utili per operare su valori del corrispondente tipo base.
  Ad esempio, le costanti static MAX_VALUE e MIN_VALUE (meno che per Boolean e Character.)

Oggetti dei tipi primitivi (2)

- Tutte queste classi forniscono il metodo di istanza toString() e il corrispondente metodo static toString(x).

```
Integer.toString(7+3);
Boolean o = new Boolean(true); . . . o.toString() . . . ;
```

- Le classi Integer e Long forniscono anche metodi statici per la conversione nella corrispondente stringa in rappresentazione binaria, ottale o esadecimale.

```
Integer.toBinaryString(7+3);
Long.toHexString(Integer.MAX_VALUE*2);
```

- Tutte le classi di interi forniscono metodi statici per convertire una stringa in un valore del corrispondente tipo base.

```
Short.parseShort("-FFF", 16); // specifica base hex
Long.parseLong("362187323110") //assume base 10;
```

- La classe Character fornisce anche metodi per determinare a quale categoria appartiene il carattere (lettera minuscola, cifra, ecc.) e per convertirlo in maiuscolo/minuscolo.

```
Character.isWhitespace('t');
Character.isLowerCase('A');
Character.toLowerCase('A');
```

Gli oggetti della classe java.lang.String

- Le stringhe sono oggetti della classe String.
- In pratica, sono involucri per i vettori di caratteri che usualmente (ad esempio in C) intendiamo come stringhe.
- La principale differenza rispetto al C è che le stringhe sono sequenze di caratteri Unicode (UTF-16 encoding).
- Come tutti gli oggetti le stringhe sono create dai costruttori della classe:

```
1 String s = new String();      // crea una stringa vuota
2 String s = new String(v);     // v e' un vettore di caratteri
3                               // crea la corrispondente stringa
4 String new = new String(old); // old e' un oggetto String
5                               //new e' una copia di old
```

- Ci sono anche costruttori per costruire stringhe a partire da sequenze di byte o caratteri (anche tenendo conto della codifica).
- Literal per rappresentare stringhe si scrivono racchiudendo sequenze di caratteri tra doppi apici ".
- Oggetti della classe String possono essere creati anche con
  String s = "The quick brown fox jumps over the lazy dog";
  Si tratta di una sorta di cast implicito da vettore di caratteri a String, che prevede la chiamata del corrispondente costruttore.
- Questa creazione implicita di oggetti è possibile anche per gli oggetti degli altri tipi primitivi. Ad esempio, Integer i = 3; Double d = 3.8;

java.lang.String: alcuni metodi (1)

Esempio di riferimento: `String greeting = "Hello";`

- Lunghezza

  ```
  public int length()
  greeting.length() == 5
  ```

  "" è la stringa vuota (lunghezza 0) ed è diversa da **null**

  ```
  String s1 = "";
  String s2 = null;
  ```

  la stringa vuota è contenuta in un oggetto, ovvero `s1 != null`

- carattere nella posizione di indice i della stringa

  ```
  public char charAt(int index)
  greeting.charAt(0) == 'H';
  greeting.charAt(4) == 'o';
  ```

- Sottostringhe

  ```
  public String substring(int beginIndex, int endIndex);
  public String substring(int beginIndex);
  "Hello".substring(1, 3) è la stringa "ell"
  "Hello".substring(2) è la stringa "llo"
  ```

java.lang.String: alcuni metodi (2)

- Test di uguaglianza

```
public boolean equals(Object anObject);
greeting.equals("Hello")
```

Notare che non si deve usare ==, perché si verificherebbe solo se si tratta di due riferimenti allo stesso oggetto.

`"Hello".substring(1, 3) == "ell"` è sempre **false**

- Comparazione lessicografica

```
public int compareTo(String anotherString)
```

restituisce 0 se le stringhe sono uguali, un valore < 0 se la stringa precede la stringa argomento nell'ordine lessicografica, un valore > 0 se la segue

`greeting.compareTo("Hello, World!")` è < 0

`greeting.compareTo("Hallo, World!")` è > 0

- Concatenazione

+ è l'operatore che concatena stringhe

```
greeting + ", World!"
```

In pratica, se un argomento di + è una stringa, anche l'altro argomento è convertito in una stringa.

```
int n = 7; String s = greeting + ", " + n;
```

s è la stringa "Hello, 7"

n + "" forza la conversione del numero 7 nella stringa "7"

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

java.lang.String: alcuni metodi (3)

- Gli oggetti sono convertiti in stringhe per mezzo del metodo toString (chiamato esplicitamente o implicitamente quando è necessaria una conversione).
  Se l'utente vuole che un oggetto di una classe da lui definita sia convertibile in una stringa, deve inserire nella classe una definizione del metodo
  ```
  public String toString()
  ```
- Per convertire stringhe in numeri possiamo usare i metodi statici parse definiti nelle classi corrispondenti ai vari tipi di numero.
  Nella classe Integer
  ```
  public static int parseInt(String s) throws NumberFormatException
  ```
  Nella classe Double
  ```
  public static double parseDouble(String s) throws NumberFormatException
  String input = "7";
  int n = Integer.parseInt(input); // restituisce l'intero 7
  ```
- Attenzione! Se le stringhe da convertire non contengono un numero, allora il metodo parse solleva una eccezione (checked) di tipo NumberFormatException, ovvero, segnala una situazione di errore.

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

Le eccezioni (1)

- Le eccezioni servono a gestire situazioni di errore o improprie dovute a
  - errori di programmazione;
    ad esempio, si tenta di accedere a un riferimento **null**, oppure si esegue una divisione e il divisore è pari a 0.
  - situazioni fuori del controllo del programmatore, perché non prevedibili o indipendenti dal programma;
    ad esempio, si vogliono leggere dei dati da un file, ma il file non esiste, è stato cancellato o è inaccessibile. Oppure, i dati nel file sono in un formato diverso da quello atteso.
- È compito del programmatore gestire le eccezioni,
  1. prendendo in considerazione la possibilità che una data eccezione possa essere sollevata;
  2. fornendo il codice per la gestione della situazione che si viene a creare nel momento in cui una eccezione è sollevata.

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

Le eccezioni (2)

- Operazioni che possono essere intraprese per gestire una eccezione (ovviamente, si possono immaginare casi più complessi):
  - ripristinare l'esecuzione del programma a partire da un nuovo stato corretto;
    ad esempio, ripristinando i dati modificati e ritornando a uno stato precedente, oppure, reinizializzando i dati e ripartendo da uno stato prefissati.
  - garantire la terminazione sicura del programma;
    ad esempio, garantendo che i dati scritti sugli stream di output siano effettivamente scritti sui corrispondenti file o inviati ai corrispondenti dispositivi e chiudendo i file aperti.
- Le eccezioni in Java sono di due tipi:

  unchecked non devono essere necessariamente gestite dal programmatore e non vengono tenute in considerazione dal compilatore;

  checked devono essere gestite dal programmatore e vengono tracciate dal compilatore:
  il compilatore segnala un errore di compilazione se il programma non dice come gestire una eccezione checked.

Eccezioni "unchecked"

- Classicamente queste eccezioni sono sollevate quando si cerca di eseguire delle operazioni non corrette a causa di un errore di programmazione o del mancato controllo della correttezza dei dati utilizzati.
- Esempio: accesso a **null** reference

```
1 String name = null;
2 int n = name.length(); // ERRORE: throws NullPointerException
```

- Dato l'elevato numero di accessi a riferimenti in un programma Java, sarebbe troppo pesante dover tener conto di tutti i possibili casi in cui si può verificare una NullPointerException. Inoltre, in molti casi, in un programma corretto si è sicuri che il riferimento usato non è **null**.
- Nel caso dell'esempio, se non si prevede un gestore dell'eccezione, il programma termina stampando una traccia dello stack dell'esecuzione.

```
1 Exception in thread "main" java.lang.NullPointerException
2  at Greeter.sayHello(Greeter.java:25)
3  at GreeterTest.main(GreeterTest.java:6)
```

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

Eccezioni "checked"

- Generalmente legate a situazioni di errore o fuori del controllo del programmatore, ovvero, che possono verificarsi indipendentemente dal modo in cui è stato progettato e scritto il programma.
- Controllare e gestire queste eccezioni fa parte della corretta realizzazione del programma, per questo il compilatore obbliga a tener conto delle eccezioni checked.
- Nei linguaggi senza eccezioni, queste situazioni devono essere gestite inserendo opportuno codice di controllo.
- Le eccezioni forniscono una maniera più flessibile e pulita di gestire queste situazioni, da interpretare come casi eccezionali rispetto al flusso normale (o principale) del codice.
- Permettono di separare la parte di codice che implementa l'algoritmo o lo specifiche del progetto, da quella che gestisce gli errori. Ad esempio, in un metodo, legati a un suo uso improprio o a una invocazione con valori dei parametri al di fuori dei valori attesi.
- Esempi di eccezioni checked nei metodi delle API:
  - NumberFormatException: può essere sollevata dai metodi parse delle classi Integer, Double, ...
  - IOException: può essere sollevata dai metodi delle classi che operano sui file o gli altri stream di input/output (ad esempio, FileReader, FileScanner, FileWriter, ...)

Oggetti dei tipi primitivi

String

Eccezioni
- Unchecked
- Checked
- Gestione
- Catch
- Finally
- Un esempio

Array e Liste
- Array
- ArrayList
- LinkedList
- ListIterator

Gestione delle eccezioni (1)

- Ci sono due modi di gestire una eccezione checked che può essere sollevata da un metodo che si sta utilizzando:
  1. passare l'eccezione a chi utilizza il metodo che contiene il frammento di codice che si sta scrivendo, dichiarando l'eccezione nel suo header;
  2. catturare e gestire l'eccezione.
- Esempio: aprire un file in lettura con il costruttore che crea un oggetto della classe FileReader può sollevare una FileNotFoundException.
  Se si usa il costruttore all'interno di un metodo read e si vuole che tale metodo, anziché catturare e gestire l'eccezione, la passi al suo chiamante possiamo scrivere:

```
1 public void read(String filename) throws FileNotFoundException
2 {
3     FileReader reader = new FileReader(filename);
4     . . .
5 }
```

- Si osservi che, per chi usa il metodo read è come se l'eccezione fosse sollevata da questo metodo.

Oggetti dei tipi primitivi

String

Eccezioni
- Unchecked
- Checked
- Gestione
- Catch
- Finally
- Un esempio

Array e Liste
- Array
- ArrayList
- LinkedList
- ListIterator

Gestione delle eccezioni (2)

- Anche il chiamante può decidere di non catturare l'eccezione facendola risalire a sua volta.
- L'eccezione può così risalire lungo lo stack delle funzioni sino all'eventuale main, che può a sua volta decidere se catturarla o dichiararla.
- Se anche il main passa l'eccezione senza catturarla, quando l'eccezione viene sollevata il programma termina tracciando lo stack dell'esecuzione.
- Un metodo può dichiarare più di una eccezione:

```
1 public void read(String filename)
2                        throws IOException, NumberFormatException;
3 public static void main(String[] args)
4                        throws IOException, ClassNotFoundException;
```

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

Catturare le eccezioni (1)

- Per catturare una eccezione occorre:
  - specificare il pezzo di codice all'interno del quale si assume possa essere sollevata l'eccezione racchiudendo il codice in un blocco preceduto dal comando **try**;
  - far seguire il blocco **try** da un blocco **catch** che specifica il tipo di eccezione che si intende catturare e il codice da eseguire nel caso tale eccezione venga catturata.

```
1 try
2 {
3     // codice che può sollevare una IOException
4 }
5 catch (IOException exception)
6 {
7     // azione di correzione dell'errore
8 }
```

- Una eccezione è un oggetto contenente le informazioni sullo stato del programma al momento in cui l'eccezione è stata sollevata.

Oggetti dei tipi primitivi
String
Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio
Array e Liste
Array
ArrayList
LinkedList
ListIterator

Catturare le eccezioni (2)

- Alcune possibili azioni correttive eseguibili nel blocco **catch** dell'esempio:
  - Segnalare un errore all'utente chiedendo un nuovo file.
  - Registrare l'errore (con le informazioni ritenute rilevanti) su di un opportuno file.
  - In un programma didattico o in una fase di test, stampare la traccia dello stack e uscire:

```
1 exception.printStackTrace();
2 System.exit(1);
```

- Per catturare più di una eccezione è necessario prevedere più blocchi **catch** in cascata

```
1  try
2  {
3      // codice che può sollevare una
4      // IOException o una NumberFormatException
5  }
6  catch (IOException exception)
7  {
8      // azione di correzione dell'errore di IO
9  }
10 catch (NumberFormatException exception)
11 {
12     // azione di correzione dell'errore di formato del numero
13 }
```

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

Il blocco "finally"

- Dopo aver eseguito un blocco **try** o **catch** può essere necessario eseguire delle azioni di cleanup.
- L'esecuzione di queste operazioni è indipendente da come termina il codice cui fanno riferimento:
  - normalmente, completando l'esecuzione del blocco **try**;
  - a seguito di una eccezione catturata, dopo aver eseguito le operazioni contenute nel corrispondente blocco **catch**;
  - a seguito di una eccezione non catturata.
- Il codice di cleanup da eseguire deve essere inserito all'interno di un blocco **finally** .
- Il blocco **finally** deve essere preceduto da un blocco **try** e da una sequenza (possibilmente vuota) di blocchi **catch**.
- Il blocco **finally** è utile e anche in assenza di eccezioni da gestire. Il suo uso permette di separare il codice principale dalle operazioni di cleanup da eseguire sempre.

Oggetti dei tipi primitivi

String

Eccezioni
- Unchecked
- Checked
- Gestione
- Catch
- Finally
- Un esempio

Array e Liste
- Array
- ArrayList
- LinkedList
- ListIterator

Un esempio di uso di "finally"

Chiudere un file.

```
1  FileReader reader = null;
2  try
3  {
4      reader = new FileReader(name);
5      ...
6  }
7  catch (FileNotFoundException e)
8  {
9      ...
10 }
11 finally
12 {
13     if (reader != null) reader.close();
14 }
```

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

Un esempio (1)

Il progetto **S06. Eccezioni** contiene degli esempi di gestione delle eccezioni.

- Le eccezioni checked vanno necessariamente gestite (catturate o passate):
  - I costruttori PrintWriter e FileWriter all'interno di writeArray possono sollevare una eccezione checked della classe IOException.
  - Questa eccezione deve essere necessariamente catturata (o passata) come fatto dal blocco **try/catch** all'interno del metodo.
- Le eccezioni unchecked non devono necessariamente essere gestite, ma si possono catturare come quelle checked.
  - Nel metodo writeArray non verifico se gli indici dell'array sono all'interno dei limiti dell'array.
  - Indici al di fuori del range dell'array causano una (unchecked) ArrayIndexOutOfBoundsException.
  - Anche se non necessario, nel metodo writeArray catturiamo anche queste eccezioni.
- In writeArray eseguo una divisione, che può causare una divisione per 0 e quindi una (unchecked) ArithmeticException
  - Questa eccezione non deve essere necessariamente gestita, ma viene implicitamente passata da writeArray al chiamante.
  - Il metodo writeArrayBis è un involucro che chiama writeArray e cattura ArithmeticException.

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

Un esempio (2)

- Nel codice di gestione delle eccezioni posso utilizzare le informazioni passate dall'istanza dell'eccezione sollevata, alcune delle quali accessibili tramite metodi dell'istanza.
  - getMessage() restituisce un messaggio che descrive l'eccezione;
  - printStackTrace() stampa sullo stream di errore (System.err) una traccia dello stack dei metodi chiamati al momento dell'eccezione.
- Notare le operazioni di cleanup nel blocco **finally** in writeArray.
  In questo caso si chiude il file se questo era stato aperto.
  Il codice nella **finally** viene eseguito anche nel caso in cui si verifichi una eccezione non catturata, ad esempio, se si verifica una ArithmeticException.
- Quando una eccezione viene catturata da una **catch**, il flusso continua normalmente dopo la **catch** come in una normale esecuzione.
  (Verificare il diverso comportamento di provaArithmeticExceptionCaught e provaArithmeticExceptionUncaught.)
  Se si vuole terminare l'esecuzione all'interno di una **catch** occorre chiamare esplicitamente la System.exit().

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

Array (1)

- Gli array permettono di memorizzare sequenze di elementi di qualsiasi tipo (riferimenti o tipi base).
- Le variabili che contengono array sono riferimenti.
- Gli array vengono creati come gli oggetti attraverso la **new**

```
int [] numbers = new int[10];
```

- Gli array hanno lunghezza fissa: non si possono inserire o eliminare elementi dagli array.
- Se si vuole cambiare la dimensione dell'array riferito da una variabile occorre allocare un nuovo array e, se necessario, copiare nel nuovo array gli elementi contenuti nel vecchio array.
- Gli elementi degli array vengono acceduti mediante indici, per mezzo dell'operatore []

```
int n = numbers[i];
```

- Gli indici di un array di n elementi variano da 0 a n-1.

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

Array (2)

- L'attributo length di un array contiene la sua lunghezza e può essere usato per controllare la variabile di iterazione:

```
1    for (int i = 0; i < numbers.length; i++) {
2        System.out.println(numbers[i]);
3    }
```

- Il costrutto **for** each fornisce un modo semplificato per costruire iterazioni su array:

```
1    for (int n: numbers) {
2        System.out.println(n);
3    }
```

La variabile n assume tutti i valori contenuti nell'array, in ordine, partendo dall'elemento nella posizione di indice 0 sino a quella di indice numbers.length−1.

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

Array (3)

- È possibile definire array di lunghezza 0

  numbers = **new int**[0];

  Notare che non si tratta di un riferimento **null**.

- Array multidimensionali

```
1   int [][] table = new int[10][2];
2   int t = table[i][j];
```

- Argomenti sulla linea di comando

  void main(String[] args

  args contiene i parametri ricevuti dal main sulla linea di comando

Array literals

- Si possono denotare degli array fornendo una sequenza di valori dello stesso tipo separati da virgole e racchiusi tra graffe:

  int [] numbers = {1,2,3,4,5,6,7,8,9,10};

  associa un array di 10 elementi a numbers e inizializza l'array ai valori della sequenza.

  numbers[0] == 1 numbers[3] == 4 numbers[9] == 10

- Gli elementi dell'array literal possono anche essere degli oggetti

```
1 Greeter[] greeters = { new Greeter("World"),
2                        new Greeter("Mario"),
3                        new Greeter("Lucia") };
```

- Possono anche essere degli array multidimensionali

```
1 int[][] table = { {1,2,3,4,5,6,7,8,9,10} ,
2                   {1,2,3,4,5,6,7,8,9,10} };
```

Oggetti dei tipi primitivi
String
Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio
Array e Liste
Array
ArrayList
LinkedList
ListIterator

ArrayList<E> (1)

- ArrayList<E> è una classe generica che permette di rappresentare array ridimensionabili di oggetti di tipo E.
- La classe è generica: definita indipendentemente dal tipo E degli oggetti.
- I generici sono stati introdotti in Java 1.5, nelle versioni precedenti ArrayList non era generica e il suo uso si basava sul sottotipaggio degli oggetti (vedremo meglio in seguito);
- per istanziare la classe occorre specificare il tipo E degli oggetti:

  ```
  ArrayList<String> countries = new ArrayList<String>();
  ```

  crea un array di 0 elementi (NB: è un oggetto con riferimento $\neq$ **null**)
- E non può essere un tipo primitivo.
- add aggiunge elementi in fondo all'array e ne aumenta la dimensione

  ```
  countries.add("Belgium");
  countries.add("Italy");
  countries.add("Thailand");
  ```
- size restituisce il numero di elementi nell'array

  ```
  countries.size() == 3
  ```

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

ArrayList<E> (2)

- get permette di accedere a un elemento in una determinata posizione

```
String country = countries.get(i);
```

  non è richiesto il cast al tipo E degli elementi;

- set cambia il valore contenuto in una determinata posizione

```
countries.set(1, "France");
    countries.get(0) == "Belgium"
    countries.get(1) == "France"
    countries.get(2) == "Thailand"
```

- L'iterazione può essere eseguita per mezzo del ciclo for each

```
1   for (String country : countries) {
2       System.out.println(country);
3   }
```

  equivalente al ciclo for tradizionale

```
1   for (int i = 0; i < countries.size(); i++) {
2       System.out.println(countries.get(i));
3   }
```

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

ArrayList<E> (3)

- Gli elementi possono anche essere inseriti e rimossi al centro dell'array

```
countries.add(1, "Germany");
    countries.get(0) == "Belgium"
    countries.get(1) == "Germany"
    countries.get(2) == "France"
    countries.get(3) == "Thailand"
countries.remove(0);
    countries.get(0) == "Germany"
    countries.get(1) == "France"
    countries.get(2) == "Thailand"
```

- Queste operazioni non sono però efficienti perché gli elementi sono memorizzati in un array all'interno di una istanza di ArrayList.
- Se il numero di add/remove che si devono eseguire è elevato conviene utilizzare LinkedList<E>.

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

LinkedList <E>

- Sono implementate mediante strutture collegate, quindi inserimento e cancellazione possono essere eseguite in modo efficiente.
- Ha più o meno gli stessi metodi di ArrayList (poiché come vedremo sono differenti implementazioni di uno stesso tipo di dato astratto).
- add aggiunge alla fine.
- get e set leggono e scrivono valori in una posizione di indice dato.
- Le operazioni più frequenti sulle liste richiedono di scorrere la lista ed eseguire delle operazioni quando si verifica una determinata condizione.
- Per scorrere una lista si possono usare gli iteratori.
- Un iteratore è una sorta di cursore che possiamo inizializzare e far scorrere in avanti lungo la lista.

Oggetti dei tipi primitivi

String

Eccezioni

Unchecked

Checked

Gestione

Catch

Finally

Un esempio

Array e Liste

Array

ArrayList

LinkedList

ListIterator

ListIterator <E> (1)

- Un iteratore è un oggetto di una determinata classe: ogni implementazione di liste o vettori ha il suo iteratore.
- L'iteratore va inizializzato creando l'oggetto (tramite il metodo di istanza listIterator di LinkedList)

```
ListIterator <String> iterator = countries.listIterator();
```

- il metodo hasNext dell'iteratore verifica se l'iteratore non ha raggiunto la fine della lista e quindi ci sono altri elementi da scorrere;
- il metodo next dell'iteratore prende il successivo elemento da scorrere e avanza l'iteratore;

```
1 while (iterator.hasNext())
2 {
3     String country = iterator.next();
4     System.out.println(country);
5 }
```

- Il ciclo for each utilizza un iteratore nascosto, permettendo un uso semplificato degli iteratori (non è necessario definire l'iteratore e farlo avanzare).

Oggetti dei tipi primitivi

String

Eccezioni
Unchecked
Checked
Gestione
Catch
Finally
Un esempio

Array e Liste
Array
ArrayList
LinkedList
ListIterator

ListIterator <E> (2)

- Gli iteratori permettono anche di inserire/cancellare facilmente elementi.
- Il metodo add dell'iteratore inserisce un elemento prima della posizione corrente dell'iteratore.
- Il metodo remove elimina l'elemento corrente (quello ottenuto con l'ultima chiamata a next).
- Il progetto sull'implementazione della liste collegate (**S05. Liste**) contiene la definizione di un iteratore.
- Questa semplice implementazione di un iteratore permette di scandire le liste ma non di eseguire efficientemente inserimenti e cancellazioni. Perché?