Sezione: Modificatori

Modificatori

Modificatori: visibilità

public	visibile da tutti
(non def.)	visibile da tutti nello stesso package
protected	visibile dalle sottoclassi
private	nascosta da tutti

Uso di metodi "di accesso":

```
public class  ACorrectClass {
    private String  aUsefulString;
    public String  getAUsefulString() {
        // "get" the value
        return aUsefulString;
    }
    private protected void setAUsefulString(String s)
{
        // "set" the value
        aUsefulString = s;
   }
}
```

Matrice degli accessi

Access Levels				
Specifier	**Class**	**Package**	**Subclass**	**World**
private	Y	N	N	N
no specifier	Y	Y	N	N
protected	Y	Y	Y	N
public	Y	Y	Y	Y

Vedi anche
http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Modificatori: final

Variabili dichiarate final sono costanti.

Metodi dichiarati final non possono essere sovrascritti

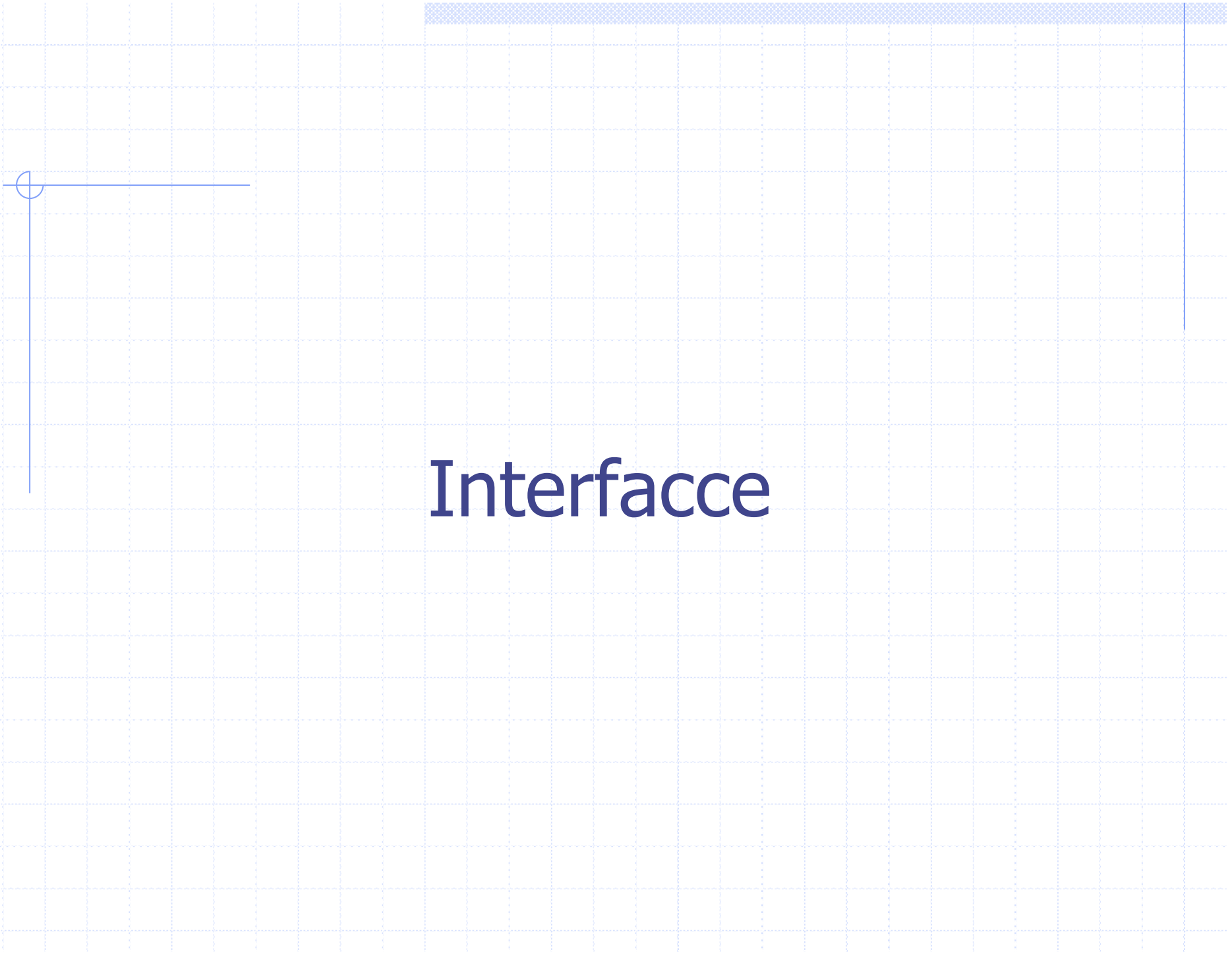
Classi dichiarate final non possono essere subclassate.

Convenzioni

I nomi delle Classi iniziano con la MAIUSCOLA

I nomi degli Oggetti iniziano con la MINUSCOLA

```
Pila p=new Pila();
```

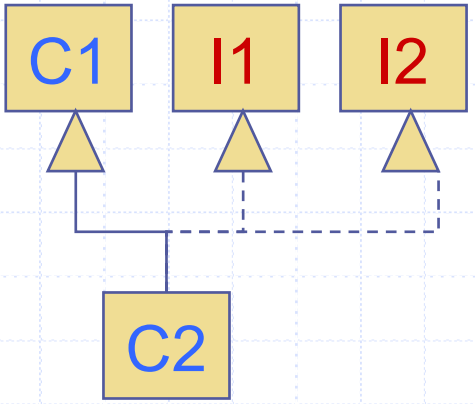

Interfacce

Interfacce

Un *interface* è una collezione di firme di metodi (senza implementazione).

Una interfaccia può dichiarare costanti.

Interfacce



Esempio di interface

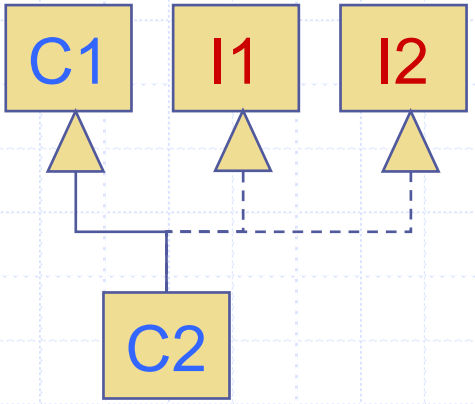
```
package strutture;
public interface Stack{
  public int estrai();
   public void insert(int z);
}
```

```
package strutture;
public class Pila implements Stack{
   …
}
```

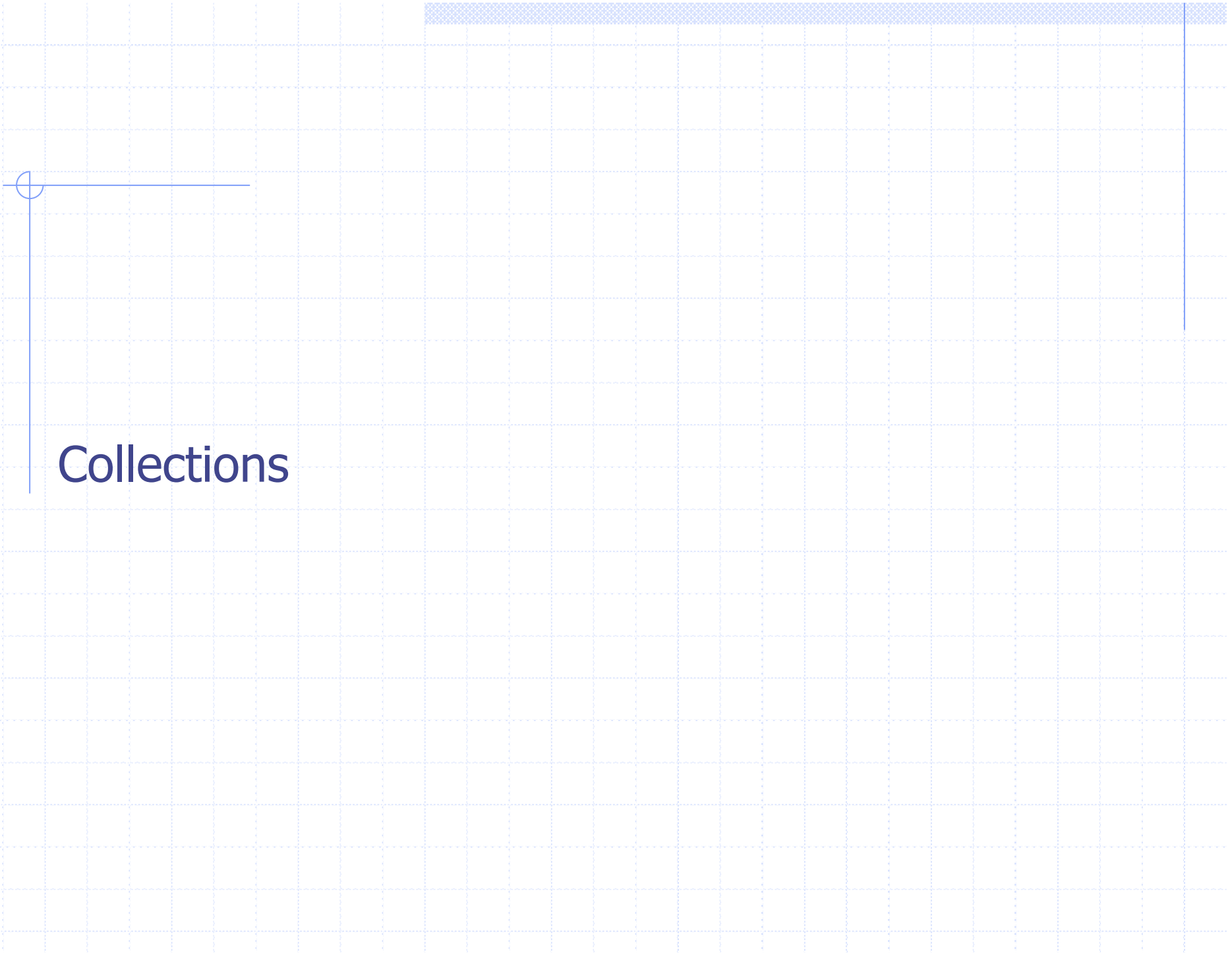
```
package strutture;
public class Coda extends Pila{
   …
}
```

Interfacce

Le interfacce possono essere usate come "tipi"



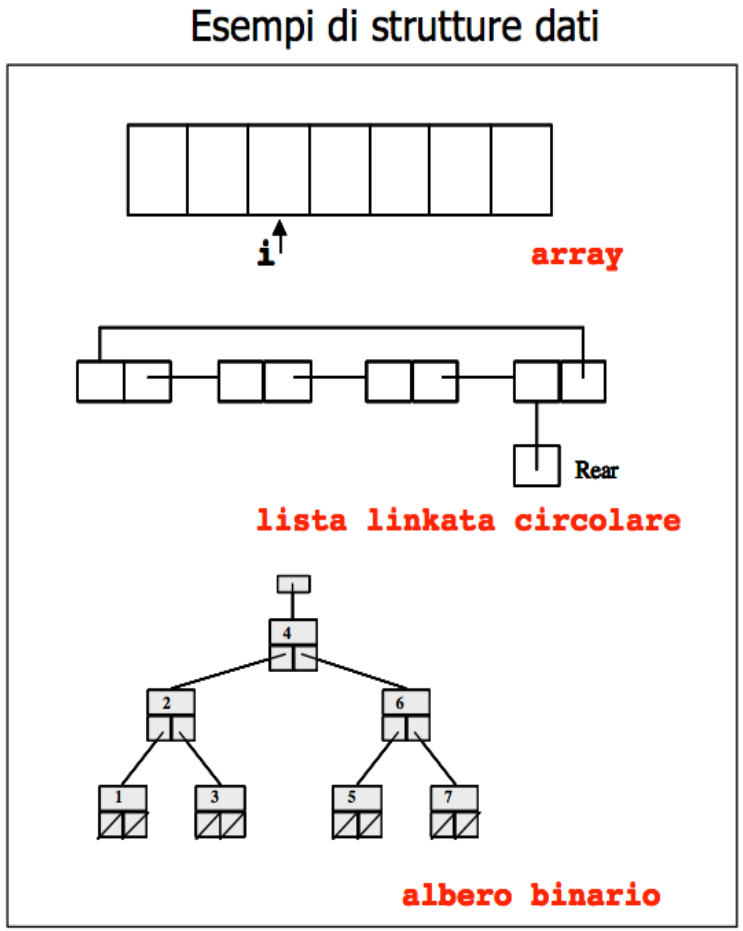

```
I1 x = new C2();
// I1 x = new I1();  NO!!
```

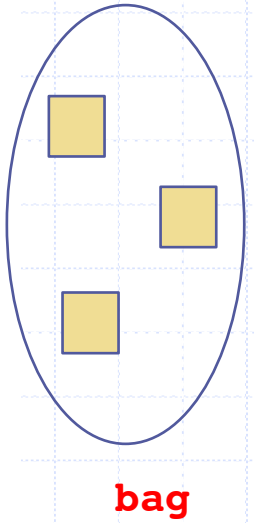

Collections

Riuso della conoscenza

Riuso del software

- **Algoritmi**
- **Pattern**

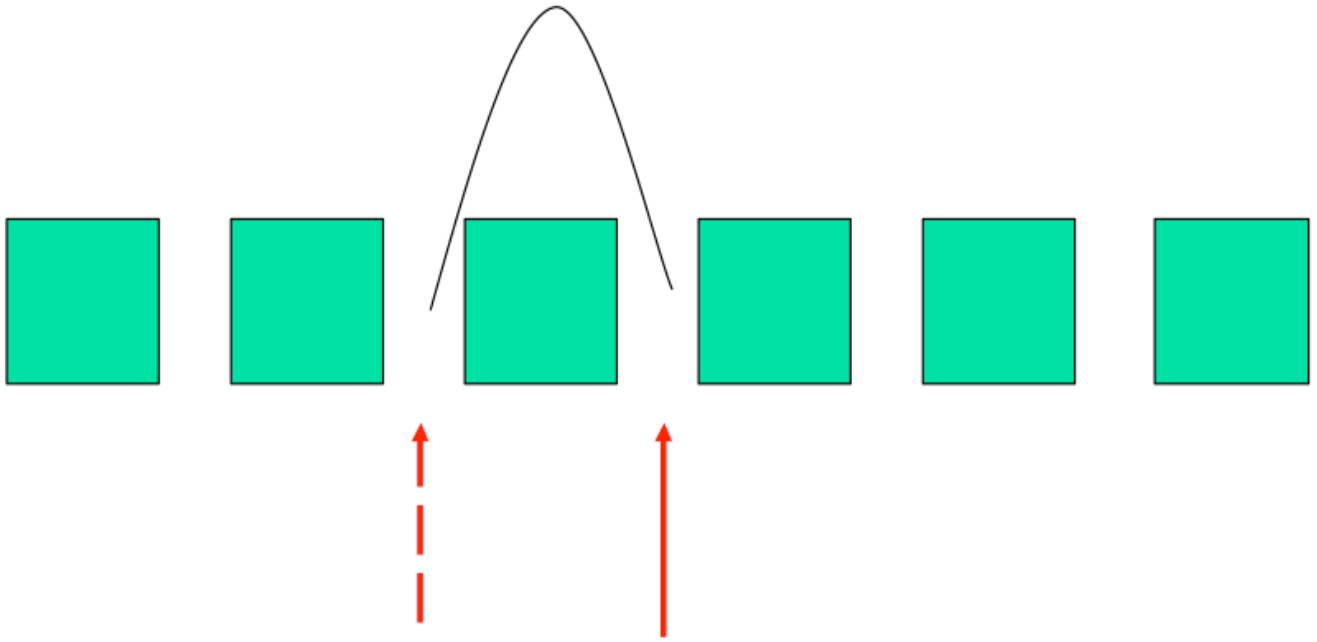




Collection: Basic operations

```
int size();
boolean isEmpty();
boolean contains(Object element);
boolean add(Object element);
boolean remove(Object element);
Iterator iterator();
```

Iterator



```
Iterator iter =  c.iterator();
obj = iter.next(); // muovi di un elemento e
                   // restituisci un riferimento associato
                   // all'elemento appena superato
```

The Iterator interface

public interface Iterator {

- boolean hasNext();
- Object next();
- void remove();

}

hasNext returns true if there are more elements in the Collection

next() returns the next element in the Collection

remove() method removes from the underlying Collection the last element that was returned by next. The remove method may be called only once per call to next, and throws an exception if this condition is violated.

Using Iterators

```
void filter(Collection x) {
    Iterator i=x.iterator();
        while (i.hasNext()) {
            if (!cond(i.next()))
                i.remove();
        }
}
```

The code is *polymorphic*: it works for *any* Collection that supports element removal, regardless of implementation. That's how easy it is to write a polymorphic algorithm under the collections framework!

Vantaggi di usare una libreria

- Ottimizza il lavoro del programmatore che si può concentrare sulle parti di contesto specifico delle sue applicazioni
- Aumenta la velocità di scrittura e la qualità del codice e d inuove API

Vantaggi di usare una libreria

- Permette la interoperabilità tra API non correlate: la maggior parte delle API accetta collections come parametri in input ed in output
- Supporta il riuso di codice: strutture dati nuove che aderiscono al collection framework sono intrinsecamente riusabili

Collections

Una collection è un oggetto che ragguppa elementi multipli (anche eterogenei) in una singola entità.

Collections sono usate per immagazzinare, recuperare e trattare dati, e per trasferire gruppi di dati da un metodo ad un altro.

Tipicamente rappresentano dati che formano gruppi “naturali”, come una mano di poker (una collection di carte), un mail folder (a collection di e-mail), o un elenco telefonico (una collection di mappe nome-numero).

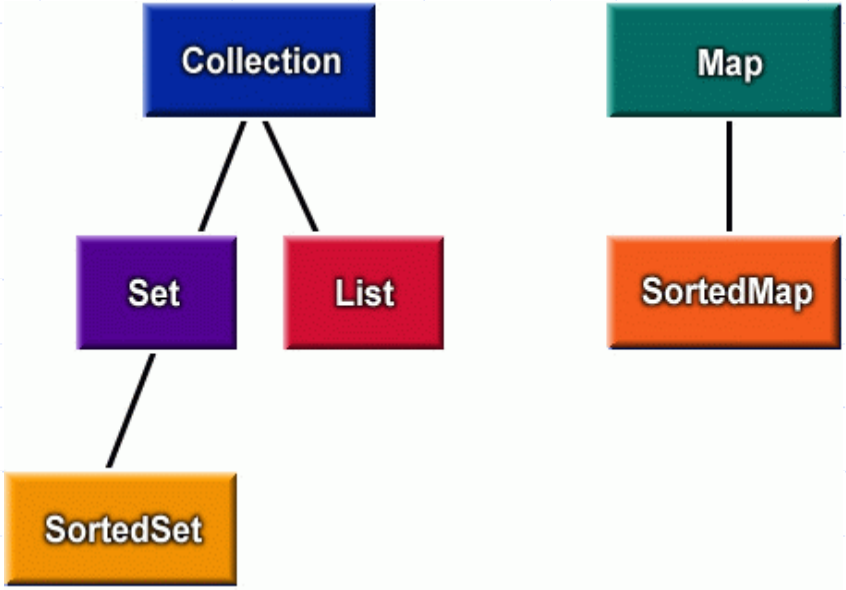
Collections Famework

Il Java Collection Framework contiene tre elementi:

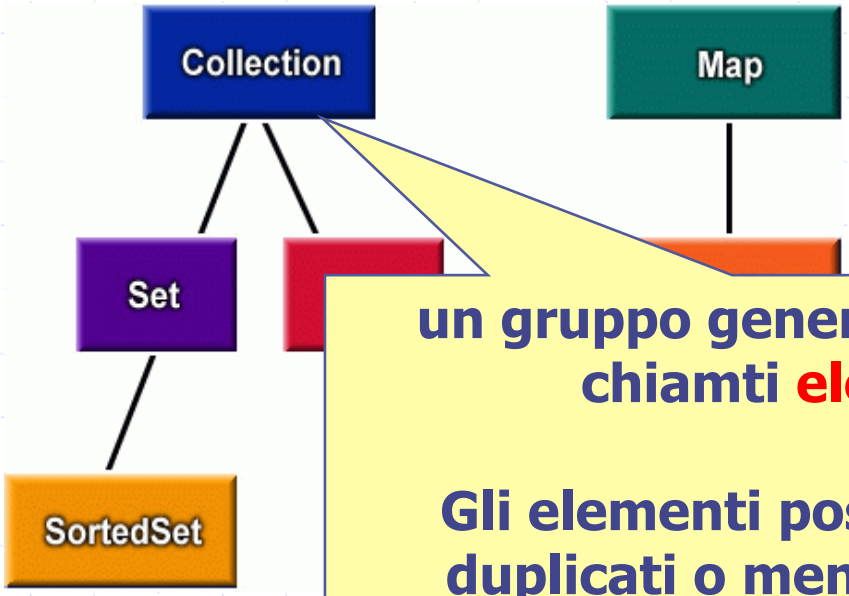
- **Interfacce**
- **Implementazioni** concrete delle interfacce precedenti;
- **Algoritmi**: metodi che implementano operazioni comuni a più strutture dati
  - Esempi: algoritmi di ricerca ed ordinamento: sort, shuffle, binarySearch, max, min...
  - Questi algoritmi sono polimorfi, nel senso che lo stesso metodo può essere usato per in diverse implementazioni concrete delle collections.

Core Collections Interfaces

Occorre importare java.util.*



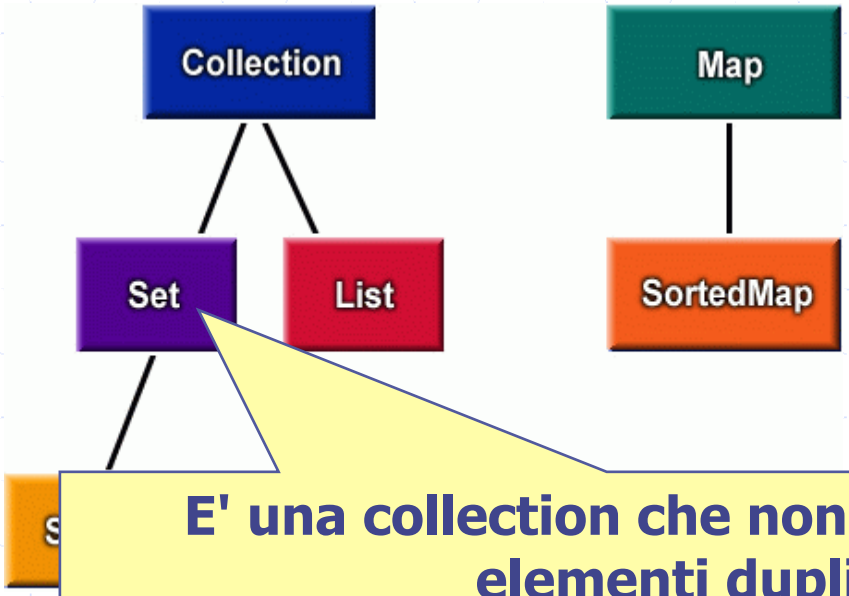
Core Collections Interfaces



un gruppo generico di oggetti chiamti elementi.

Gli elementi possono essere duplicati o meno, ordinati o disordinati etc..

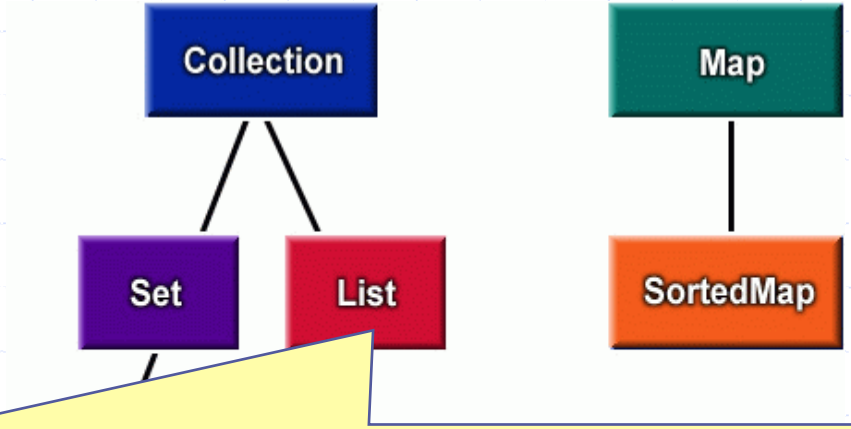
Core Collections Interfaces



E' una collection che non puo`contenere elementi duplicati

Esempi concreti: l'insieme dei processi che girano su un computer, una mazzo di carte da briscola…

Core Collections Interfaces



List è una collection ordinata che puo`contenere elementi duplicati.
Normalmente fornisce all'utente controllo nell'inserimento.
L'utente accede agli elementi attraverso un indice (posizione).
Esempi di implementazione: vettori, liste linkate.
Esempi concreti: Mazzo di carte da Scala 40

Differenze tra Set e List

	Elementi duplicati	Elementi accessibili con un indice	Gli elementi mantengono l'ordine di inserimento
Set	NO	NO	NO
List	SI	SI	SI

Collection: Basic operations

```
int size();
boolean isEmpty();
boolean contains(Object element);
boolean add(Object element);
boolean remove(Object element);
Iterator iterator();
```

Collection: basic operations

- The add method is defined generally enough so that it makes sense for collections that allow duplicates as well as those that don't. It guarantees that the Collection will contain the specified element after the call completes, and returns true if the Collection changes as a result of the call.
- The remove method is defined to remove *a single instance* of the specified element from the Collection, assuming the Collection contains the element, and to return true if the Collection was modified as a result.

Collection: bulk operations

```
// Bulk operations
boolean containsAll(Collection c);
boolean addAll(Collection c);
boolean removeAll(Collection c);
boolean retainAll(Collection c);
void clear();
// Array Operations
Object[] toArray();
Object[] toArray(Object a[]);
```

Core Collections Interfaces

		Implementations			
		Hash Table	Resizable Array	Balanced Tree	Linked List
Interfaces	Set	HashSet		TreeSet	
	List		ArrayList		LinkedList
	Map	HashMap		TreeMap	

Fondamenti di Java

Implementazione di Pila e Coda usando le Collection

Number

```
package structures;
class Number {

    private int n;

    Number(int n) {
        this.n = n;
    }

    int getInt() {
        return n;
    }

    void setInt(int n) {
        this.n = n;
    }
}
```

Stack

```
package structures;
import java.util.*;
public abstract class Stack extends
LinkedList {

    public void inserisci(int x) {
        Number n = new Number(x);
        this.add(n);
    }

    abstract public int estrai();
}
```

Coda

```
class Coda extends Stack {

    public int estrai() {
        Number x = null;
        Iterator iter = this.iterator();
        if (iter.hasNext()) {
            x = (Number) iter.next();
            iter.remove();
        } else {
            System.out.println("Tentativo di
estrarre                    da una Coda vuota");
            System.exit(1);
        }
        return x.getInt();
    }
}
```

Coda

```
class Pila extends Stack {

    public int estrai() {
        Number x = null;
        Iterator iter = this.iterator();
        while (iter.hasNext()) {
            x = (Number) iter.next();
        }
        if (x == null) {
            System.out.println("Tentativo di
estrarre                    da una Pila vuota");
            System.exit(1);
        }
        iter.remove();
        return x.getInt();
    }
}
```

main

```
public static void main(String[] args) {
    Stack s=new Coda(); // Stack s=new Pila();
    s.inserisci(1);
    s.inserisci(2);
    s.inserisci(3);
    for (int k=0;k<=4;k++){
       int v=s.estrai();
        System.out.println(v);
    }
 }
```

1
2
3
Tentativo di estrarre da una Coda vuota

3
2
1
Tentativo di estrarre da una Pila vuota

Stack

```
package structures;
import java.util.*;
public abstract class Stack extends ArrayList
{

    public void inserisci(int x) {
        Number n = new Number(x);
        this.add(n);
    }

    abstract public int estrai();
}
```

e se avessi esteso HashSet?

Javadoc

Input

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The
name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param url an absolute URL giving the base location of the image
 * @param name the location of the image, relative to the url
argument
 * @return the image at the specified URL
 * @see Image
 */
public Image getImage(URL url, String name) {
   try { return getImage(new URL(url, name));
   } catch (MalformedURLException e) { return null; }
}
```

Output

getImage

public Image **getImage**(URL url, String name)

Returns an Image object that can then be painted on the screen. The url argument must specify an absolute URL. The name argument is a specifier that is relative to the url argument. This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

url - an absolute URL giving the base location of the image

name - the location of the image, relative to the url argument

Returns:

the image at the specified URL

See Also:

Image

Tags

Include tags in the following order:

@author (classes and interfaces only, required)
@version (classes and interfaces only, required.)
@param (methods and constructors only)
@return (methods only)
@exception (@throws is a synonym added in Javadoc 1.2)
@see (additional references)
@since (since what version/ since when is it available?)
@serial (or @serialField or @serialData)
@deprecated (why is deprecated, since when, what to use)

Documentation generation

To generate the html documentation, run javadoc followed by the list of source files, which the documentation is to be generated for, in the command prompt (i.e. *javadoc [files]*).

javadoc also provides additional options which can be entered as switches following the javadoc command (i.e. *javadoc [options] [files]*).

javadoc options

Here are some basic javadoc options:

- ***-author*** **- generated documentation will include a author section**
- ***-classpath [path]*** **- specifies path to search for referenced .class files.**
- ***-classpathlist [path];[path];...;[path]*** **- specifies a list locations (separated by ";") to search for referenced .class files.**
- ***-d [path]*** **- specifies where generated documentation will be saved.**
- ***-private*** **- generated documentation will include private fields and methods (only public and protected ones are included by default).**
- ***-sourcepath [path]*** **- specifies path to search for .java files to generate documentation form.**
- ***-sourcepathlist [path];[path];...;[path]*** **- specifies a list locations (separated by ";") to search for .java files to generate documentation form.**
- ***-version*** **- generated documentation will include a version section**

Examples

Basic example that generates and saves documentation to the current directory (c:\MyWork) from A.java and B.java in current directory and all .java files in c:\OtherWork\.

c:\MyWork> javadoc A.java B.java c:\OtherWork*.java

More complex example with the generated documentation showing version information and private members from all .java files in c:\MySource\ and c:\YourSource\ which references files in c:\MyLib and saves it to c:\MyDoc.

c:\> javadoc -version -private -d c:\MyDoc
-sourcepathlist c:\MySource;c:\YourSource
-classpath c:\MyLib

More info

http://java.sun.com/j2se/javadoc/writingdoccomments/index.html

The javadoc tool does not directly document anonymous classes -- that is, their declarations and doc comments are ignored. If you want to document an anonymous class, the proper way to do so is in a doc comment of its outer class