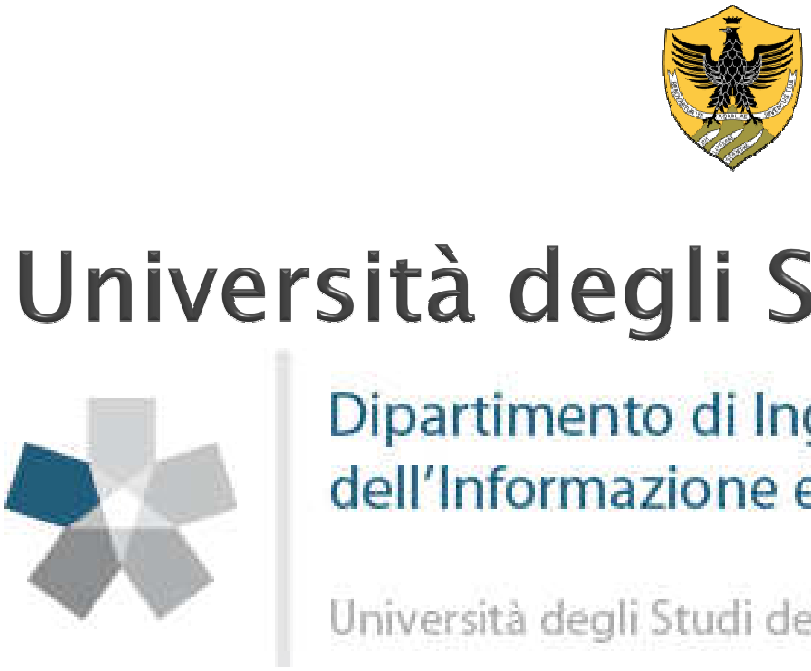

Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Università degli Studi dell'Aquila

Corso di Algoritmi e Strutture Dati con Laboratorio

Tipi di dato pila e coda

Tipo di dato Pila (Stack)

- Uno **stack** è una collezione di elementi (tipo di dato astratto) che supporta le seguenti operazioni tipiche:
  - `push, pop, peek o top, isEmpty`
  - `eventualmente isFull`
- Disciplina di accesso: l'accesso agli elementi avviene secondo l'ordine inverso di inserimento (politica **LIFO – last in first out**)
- Esempi pratici sono le pile di piatti o le pile di giornali, che sottendono l'idea che quando si pone un piatto/giornale nella pila lo si metta in cima, e che quando lo si preleva, si prelevi analogamente quello in cima

Tipo di dato Stack

tipo `Stack`:
dati: una sequenza S di n elementi.
operazioni:

`isEmpty()` → *result*
restituisce `true` se S è vuota, e `false` altrimenti

`push`(*elem e*)
aggiunge *e* come ultimo elemento di S

`peek()` → *elem // altrimenti riferita come top()*
restituisce l'ultimo elemento di S (senza eliminarlo da S)

`pop()` → *elem*
elimina da S l'ultimo elemento e lo restituisce

Tipo di dato Pila

- Il termine stack viene usato in informatica in modo più specifico in diversi contesti:
- lo stack è un elemento dell'architettura dei moderni processori, e fornisce il supporto fondamentale per l'implementazione del concetto di subroutine (vedi call stack, ricorsione)
- le macchine virtuali di quasi tutti i linguaggi di programmazione ad alto livello usano uno stack dei record di attivazione per implementare il concetto di subroutine (generalmente, ma non necessariamente, basandosi sullo stack del processore)
- la memoria degli automi a pila dell'informatica teorica è uno stack

Tipo di dato Pila

Esempi

- Verificare il bilanciamento delle parentesi in espressioni e programmi

```
abc{defg{ijk}{l{mn}}op}qr   (true)
abc{def}}{ghij{kl}m         (false)
abc{def}{ghij{kl}m          (false)
```

- Riconoscere stringhe palindrome

```
abcdcba
```

- Valutare espressioni postfisse

```
2 3 4 + *
```

Stacks in Java

- La classe `stack` nel package `java.util` dovrebbe essere evitata poiché è una sottoclasse di **`Vector`** e perciò consente l'esecuzione di operazioni non-stack.

Approfondimento: Vector<E> vs ArrayList<E>

- Le classi generiche **Vector<E>** e **ArrayList<E>** sono sostanzialmente equivalenti, ma:
- I metodi di **Vector<E>** sono **sincronizzati**, mentre quelli di **ArrayList<E>** non lo sono. Quindi se il programma è **concorrente** (cioè usa il **multi-threading** di Java) è opportuno usare **Vector<E>**, altrimenti converrebbe **ArrayList<E>** perché più efficiente.
- **Vector<E>** fornisce, con opportuni metodi e costruttori, un controllo maggiore sulla **capacità**, cioè la dimensione dell'array soggiacente.
- Per motivi storici, **Vector<E>** fornisce più metodi con nomi diversi per manipolare gli elementi di un vettore.

Vector<E> vs ArrayList<E>

- I costruttori `di` **`Vector<E>`** permettono di specificare la capacità iniziale del vettore (**`initialCapacity`**) e il valore da usare per aumentare la capacità (**`capacityIncrement`**) quando necessario.
  - Se (**`capacityIncrement`** == 0), il nuovo array avrà capacità doppia rispetto all'attuale.
- I costruttori di **`ArrayList<E>`** permettono di specificare solo la capacità iniziale del vettore.

Vector<E> vs ArrayList<E>

- **Vector<E>**

```
/* crea un vettore vuoto, con i parametri specificati */
Vector (int initialCapacity, int capacityIncrement)
/* default: capacityIncrement==0 */
Vector (int initialCapacity)
/* default: initialCapacity==10 e capacityIncrement==0 */
Vector ()
```

- **ArrayList<E>**

```
/* crea un vettore con la capacità iniziale indicata */
ArrayList (int initialCapacity)
/* crea un vettore vuoto; */
ArrayList ()
```

L'interfaccia Stack

- La seguente interfaccia definisce le operazioni di interesse di una pila

  Rif. Stack.java

```
public interface Stack<T> {
  void push(T item);
  T pop();
  T peek();
  int size();
  boolean isEmpty();
}
```

Implementazioni

- Implementazione semplice basata su LinkedList: lo stack “delega” banalmente alla lista!

  Rif. SimpleStack.java
- Implementazione basata su array (tecnica del raddoppiamento-dimezzamento)

  Rif. ArrayStack.java
- Implementazione basata su liste collegate semplici

  Rif. LinkedStack.java

Javadoc (breve introduzione)

- Javadoc è uno strumento che permette di documentare i sorgenti di un programma all'interno dei sorgenti stessi.
- Anziché scrivere la documentazione di un programma in un file separato, il programmatore inserisce nel codice sorgente dei commenti in un formato particolare.
- Tali commenti vengono estratti dal programma javadoc che li converte in un formato più semplice per la consultazione (HTML o PDF, ad esempio)

Javadoc (breve introduzione)

- In generale, si dovrebbe commentare tutto ciò che utile per far comprendere il proprio codice ai colleghi e, per gli aspetti più generali, agli utenti.
- Un commento Javadoc è pensato solo per descrivere le funzionalità o i principi di un package/classe/interfaccia/metodo.
- Un commento Javadoc non è pensato per "spiegare" pezzi di codice. Questi commenti, anche se sono in formato Javadoc, non sono mai estratti dal comando javadoc.
- La documentazione minima dovrebbe comprendere la descrizione di ciascun **package, classe, interfaccia, metodo pubblico, attributo pubblico.**

Javadoc (breve introduzione)

In generale un commento Javadoc è un testo HTML racchiuso tra i tag /** e */.

Esempio Javadoc

```
/**
 * Questo &egrave; un commento <em>Javadoc</em>.
 * Gli spazi e gli asterischi a inizio riga
 * <strong>sono</strong> sempre ignorati.
 */
```

L'effetto è il seguente:

Questo è un commento *Javadoc*. Gli spazi e gli asterischi a inizio riga **sono** sempre ignorati.

Javadoc (breve introduzione)

Un commento Javadoc:

- è sempre posto subito prima della dichiarazione della **classe, interfaccia, metodo, attributo.**
- deve descrivere in modo sintetico lo scopo dell'oggetto che si sta documentando.
- può contenere tag HTML per aiutare la formattazione, anche se si sconsiglia di adottare tag di presentazione come <b>, <i>, ecc in favore di tag di struttura come , , ecc
- può contenere tag speciali (prossime slide) che aiutano a organizzare meglio il commento.
- Un commento Javadoc non deve essere una lezione. Deve aiutare a comprendere e non far perdere tempo.

Javadoc (breve introduzione): un esempio completo

```
/**
 * 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 location of the image
 * @param  name the location of the image
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name);
```

Javadoc (breve introduzione)

Il commento precedente è composto da:

- Una prima parte composta da un paragrafo che riassume lo scopo del metodo.
- Eventuali paragrafi successivi sono suddivisi da <p>.
- Una riga vuota che separa la parte descrittiva dalla parte dei tag Javadoc. Deve esserci solo 1 riga vuota in tutto.
- Una seconda parte in cui sono inseriti tag Javadoc (ad esempio, @param, spiegati nelle prossime slide) che identificano le componenti dell'oggetto che si sta commentando.

Javadoc (breve introduzione): i tag

- Formato generale di un tag: **@name comment** dove nome specifica quale tipo di informazione si sta dando e il commento è l'informazione.
- Esempio: **@author William Shakespeare**
- Ogni tag deve essere su una riga nuova.
- I commenti possono estendersi su più righe, ma non ci devono essere righe vuote!

Javadoc (breve introduzione): i tag per le classi

I tag che si devono mettere sono:

@author chi ha scritto il programma. Formato: Nome Cognome (no nick!)
Se ci sono più autori, si mettono più tag su righe separate, uno di seguito all'altro.

Tag minimo per le classi

```
/** ...
  *
  * @author William Shakespeare
  * @author Christopher "Kit" Marlowe
  */
  public class Drama {...}
```

Javadoc (breve introduzione): i tag per i metodi

I tag che si devono mettere sono (in ordine di apparizione):

@param <nome parametro> breve descrizione del parametro. Se ci sono più parametri, rispettare l'ordine con cui sono dichiarati.

@return breve descrizione di ciò che è tornato dal metodo. Esempio: `@return true if the value was found in the array, false otherwise.`

@exception <nome eccezione> descrizione delle circostanze che determinano il lancio dell'eccezione.

Javadoc (breve introduzione): i tag per i metodi

Tag minimo per i metodi

```
/**
 * Gets <code>item</code>.
 *
 * @param item searched element
 * @return the node that contains <code>item</code>
 *         if it is present, null otherwise.
 * @exception NullPointerException if <code>item</code>
 *                                 is null.
 */
BinaryNode get(Comparable item);
```

Javadoc (breve introduzione): la compilazione

- Il comando per estrarre la documentazione è javadoc presente in tutte le Java Development Kit (JDK)
- Il comando javadoc può produrre la documentazione in diversi formati, ciascuno dei quali può richiedere dei file di istruzioni speciali (i cosidetti doclet)
- **Il formato standard è HTML e non richiede nessun file ausiliario.**

Javadoc (breve introduzione): la compilazione

- Si supponga di voler ottenere la documentazione di un proprio package (`labASD`) i cui sorgenti sono nella directory `$HOME/labASD/src/`.
- Si supponga di voler salvare la documentazione (che sarà composta da più file HTML) nella directory `$HOME/labASD/doc/`.
- Si supponga di voler estrarre la documentazione di tutti i metodi/attributi (anche i privati).
- Ci si posizioni nella directory `$HOME/labASD/`.
- Il comando da lanciare è il seguente:

```
javadoc -private -sourcepath ./src -d ./doc
labASD
```

Javadoc (breve introduzione): la compilazione

- Se si vuole eseguire javadoc soltanto su files .java selezionati, è necessario indicare i files sorgenti separati da spazi oppure usare un asterisco (*) come wildcard.
- Far attenzione ad includere i path completi
- Esempio (windows):

```
C:\> javadoc -d [path to javadoc
  destination directory] [source filenames]
C:\> javadoc -d .\doc .\src\*.java
```

Tipo di dato Coda

- Una **coda** è una collezione di elementi (tipo di dato astratto) che supporta le seguenti operazioni tipiche:
  - `Enqueue (add), dequeue (remove), peek, isEmpty, eventualmente isFull`
- Disciplina di accesso: l'accesso agli elementi avviene secondo l'ordine di inserimento (politica **FIFO – first in first out**)
- Esempi pratici: una coda che in un paese civile si fa per ottenere un servizio (es: una coda di persone alla cassa di un supermercato), una coda di ordini da evadere, una coda di file da stampare, ...

Tipo di dato Coda

Questo tipo di struttura dati:

- è molto utilizzata in Informatica, ad esempio nella gestione delle operazioni da eseguire da parte di un sistema operativo
- è fondamentale nelle telecomunicazioni, in particolare nelle reti a commutazione di pacchetto, dove descrive la gestione dei pacchetti in attesa di essere trasmessi su un collegamento.

Tipo di dato Queue

tipo `Queue`:
dati: una sequenza S di n elementi.
operazioni:

`isEmpty()` → *result*
  restituisce `true` se S è vuota, e `false` altrimenti

`add`(*elem e*)
  aggiunge *e* come ultimo elemento di S

`peek()` → *elem // altrimenti riferita come first()*
  restituisce il primo elemento di S (senza eliminarlo da S)

`remove()` → *elem*
  elimina da S il primo elemento e lo restituisce

L'interfaccia Coda

- La seguente interfaccia definisce le operazioni di interesse di una coda

  Rif. Queue.java

```
public interface Queue<T> {
  public boolean isEmpty();
  public boolean add (T e);
  public T peek();
  public T remove();
  public int size();
}
```

Implementazioni

- Implementazione semplice basata su LinkedList: la coda “delega” banalmente alla lista!

  Rif. SimpleQueue.java
- Implementazione basata su array a dimensione fissa (buffer circolare)

  Rif. BoundedQueue.java
- Implementazione basata su liste collegate semplici

  Rif. LinkedQueue.java

L'interfaccia Queue<E> (JCF)

```
public interface Queue<E> extends Collection<E> {
  E element();
  boolean offer(E e);
  E peek();
  E poll();
  E remove();
}
```

- Le classi `LinkedList<E>` e `PriorityQueue<E>` implementano l'interfaccia `Queue<E>`

L'interfaccia Queue<E> (JCF)

Ogni metodo esiste in due forme:

1. Una solleva un'eccezione se l'operazione fallisce
2. L'altra restituisce un valore speciale se l'operazione fallisce

	Throws exception	Returns special value
Insert	add(e)	offer(e)
Remove	remove()	poll()
Examine	element()	peek()