Stack

Corso: Strutture Dati

Docente: Annalisa De Bonis

Tipi di Dati Astratti (TDA)

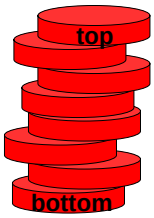
- Un tipo di dato astratto è un'astrazione di una struttura dati:
- Per definire un tipo di dato astratto occorre specificare:
  - I dati immagazzinati
  - Le operazioni supportate
  - La condizioni di errore associate alle operazioni

Tipi di Dati Astratti (TDA)

- Esempio: Un TDA che rappresenta un sistema di prenotazione di esami universitari.
  - Dati immagazzinati: prenotazioni esami
  - Operazioni supportate:
    - prenota(esame, data, studente)
    - cancella(esame,studente)
    - modifica(esame,vecchiaData,nuovaData,studente)
  - Condizioni di errore:
    - Prenota un esame inesistente
    - Prenota un esame per una data inesistente
    - Cancella una prenotazione inesistente
    - Modifica una prenotazione inesistente

Il TDA Stack (pila)

- Lo Stack è un TDA che immagazzina oggetti arbitrari
- Inserimenti e cancellazioni avvengono secondo lo schema LIFO (last-in-first-out)
  - Un nuovo oggetto viene inserito in cima (top) allo stack
  - L'elemento cancellato è sempre quello che si trova in cima allo stack



Il TDA Stack

- Operazioni supportate:
  - push(x): inserisce l'elemento x allo stack
  - pop(): cancella e restituisce in output l'ultimo elemento inserito
  - top(): restituisce l'ultimo elemento senza cancellarlo
  - size(): restituisce il numero di elementi dello stack
  - isEmpty(): restituisce **true** o **false** a seconda che lo stack sia vuoto o meno

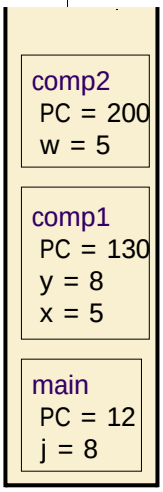
Il TDA Stack

- Esempi di applicazioni:
  - Storia delle pagine web visitate da un browser
  - Sequenza dei record di attivazione durante l'esecuzione di un programma
  - Struttura dati ausiliaria di un algoritmo

Lo stack dei metodi nella JVM

- La Java Virtual Machine (JVM) tiene traccia della catena di metodi attivi con uno stack
- Quando viene invocato un metodo JVM fa il push nello stack del frame contenente
  - **Parametri, variabili locali e valori di ritorno**
  - **Program counter**
- Quando l'esecuzione di un metodo termina, il suo frame viene estratto (pop) dallo stack e il controllo passa al metodo al top dello stack.
- Permette la ricorsione

```
main() {
   int j = 8;
    …
12  comp1(j);
  …  }
comp1(int y) {
  int x;
  x = y-3;
130 comp2(x);
  …  }
comp2(int w) {
200 …  }
```

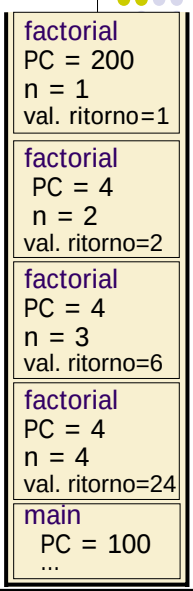


Lo stack dei metodi nella JVM

- Esempio dell'uso dello stack dei metodi nell'esecuzione di un metodo ricorsivo

```
main() {
…
100 factorial(4);
….
}

factorial(int n) {
   if(n<2)
200      return 1;
   else
202      return n*fattoriale(n-1);
}
```



Interfaccia dello Stack (non generica)

- Corrisponde al TDA Stack
- Richiede la definizione di EmptyStackException
- NB: per ora non usiamo i tipi generici

```
public interface Stack{

  public int size();

  public boolean isEmpty();

  public Object top()
      throws EmptyStackException;

  public void push(Object o);

  public Object pop()
       throws EmptyStackException;
}
```


Eccezioni

- Le operationi pop e top non possono essere eseguite se lo stack è vuoto
- Il tentativo di eseguire un'operazione di pop o di top su uno stack vuoto provoca (throw) un'eccezione di EmptyStackException

Stack implementati con array

- Gli elementi vengono aggiunti da sinistra verso destra
- Una variabile tiene traccia dell'indice dell'elemento al top

Uno stack di n elementi richiede spazio O(n)

Stack implementati con array

- Limiti dell'implementazione con array (non del TDA)
  - La dimensione massima dello stack deve essere stabilita a priori e non può essere cambiata
  - L'array può riempirsi totalmente
  - Un'operazione di push su uno stack pieno provoca (throw) una FullStackException

Stack in Java: implementazione con array (classe non generica)

```
public class ArrayStack  implements Stack {

  // contiene gli elementi dello stack
  private Object S[ ];

  // indice del top
  private int top = -1;

  // costruttore
  public ArrayStack(int capacity) {
     S = new Object[capacity]);
   }
```


Stack in Java: implementazione con array (classe non generica)

```
public Object pop()
      throws EmptyStackException {
   if isEmpty()
      throw new EmptyStackException
        ("Stack vuoto:  pop non possibile");
   Object temp = S[top];
   // facilita la garbage collection
   S[top] = null;
   top = top – 1;
   return temp;
 }
```

Stack in Java: implementazione con array (classe non generica)

```
  public Object top()
        throws EmptyStackException {
     if isEmpty()
        throw new EmptyStackException  ("Stack vuoto:  pop non possibile");
     return( S[top]);
        }
   public void push(Object element) throws FullStackException {
     if (size() == S.length)
        throw new FullStackException("Lo stack è pieno");
      S[++top] = element;
                                              }
}
```


Implementazione dell'eccezione EmptyStackException

```
public class EmptyStackException
    extends RuntimeException {
  public EmptyStackException(String errore) {
    super(errore);
  }
}
```

Stack basati su array dinamici

- Quando l' array è pieno, la push rimpiazza l'array con uno piu` grande
- Quanto più grande deve essere l'array?
  - Strategia incrementale: aumenta la dimensione di una costante c
  - Strategia del raddoppio: raddoppia la dimensione

La classe non generica ArrayStack

- Abbiamo definito Stack in modo che funzioni qualsiasi sia il tipo degli elementi memorizzati
  - I metodi prendono in input e restituiscono elementi di tipo Object
- Inconvenienti:
  - Lo stack può contenere elementi di tipo diverso
  - Quando si invocano top e pop occorre fare il cast al tipo originale dell'elemento restituito
    - Esempio : `String nome = (String) S.pop();`

Tipi generici

- Classi e interfacce generiche
- Permettono di definire una classe o un'interfaccia in termini di un insieme di *parametri formali di tipo*
  - Quando si istanzia un oggetto della classe si specificano i *tipi attuali*

Interfaccia generica dello Stack

```
public interface Stack<E> {

  public int size();

  public boolean isEmpty();

  public E top()
      throws EmptyStackException;

  public void push(E o);

  public E pop()
      throws EmptyStackException;
}
```

Stack in Java: implementazione con array (classe generica)

```
public class ArrayStack<E>
   implements Stack<E> {

   // contiene gli elementi dello
stack
   private E S[ ];

   // indice del top
   private int top = -1;

   // costruttore
   public ArrayStack(int capacity) {
   S = (E[ ])new Object[capacity]);
    }
```

```
   public E pop()
        throws EmptyStackException {
     if isEmpty()
        throw new EmptyStackException
          ("Stack vuoto:  pop non possibile");
     E temp = S[top];
     // facilita la garbage collection
     S[top] = null;
     top = top – 1;
     return temp;
   }

...
```


Stack in Java: implementazione con array (classe generica)

```
public void push(E o)  throws FullStackException{
If(size()==S.length)
  throw new FullStackException(…)
S[++top]=o;
}
```

Uso della classe generica ArrayStack

- Stack <String>S = new ArrayStack<String>(10)
  - Crea un'istanza di ArrayStack per elementi di tipo String
- String stringa = S.top();
  - Assegna alla variabile stringa l'elemento al top di S

Vantaggi

- Evitano un gran numero di cast
  - Non occorre effettuare il cast al tipo originale degli elementi restituiti dai metodi della classe
- Consentono un controllo sull'omogeneità degli elementi di un contenitore
  - Non è possibile definire un contenitore che contiene elementi di tipi diversi
  - Ci sono maggiori controlli a tempo di compilazione e di conseguenza il codice è più robusto

Omogeneita` degli elementi del contenitore

```
/*******   tipo non generico ArrayStack   *******/

ArrayStack S = new ArrayStack();
S.push(5);
int x = (Integer)S.pop();
S.push("3");   // ok


/*******   tipo generico ArrayStack   *******/

ArrayStack<Integer> S =  new ArrayStack<Integer>();
S.push(5);
int x = S.pop();
S.push("3"); // errore: non posso inserire stringhe
```


Esempio di classe con più parametri di tipo: la classe Pair

```
public class Pair<K, V>
 {
 K key; V value;
public void set(K k, V v)
 { key = k; value = v; }
 public K getKey() { return key; }
 public V getValue() { return value; }
 public String toString()
  { return "[" + getKey() + ", " + getValue() + "]"; }
 }
```

Una funzione main che usa Pair

```
public static void main (String[] args) {
  Pair<String,Integer> pair1 = new Pair<String,Integer>();
  pair1.set("altezza", 10);
 String key1= pair1.getKey();  //non serve il cast a String
 Integer value1= pair1.getValue();  //non serve il cast ad Integer.
  …
}
```


Esercizio

- Completare l'implementazione della classe generica ArrayStack
- Modificare la classe ArrayStack in modo che se lo stack è pieno venga allocato dello spazio sufficiente a contenere nuovi elementi e non venga lanciata l'eccezione **FullStackException**

Esercizio

- Scrivere un programma che utilizza uno stack per invertire una stringa.

Algoritmo che controlla se in una stringa le parentesi tonde sono ben accoppiate

Input: stringa $s=s_0,..,s_{n-1}$ contente parenteis tonde

Output: true se s è ben parentesizzata; false altrimenti

```
Sia S uno stack
for i=0 to n-1 do
  if s_i ='(' then
        S.push(s_i)
  else if s_i =')' then
        if S.isEmpty() then
              return false //non c'è la corrispondente
                           //parentesi aperta
        else S.pop()
  if S.isEmpty() then return true
  else  return false //una o più parentesi aperte per cui
                     //non c'è la corrispondente parentesi chiusa
```

Come modificare l'algoritmo perché funzioni in presenza di diversi tipi di parentesi

- Se si incontra una parentesi aperta (di qualsiasi tipo essa sia) si fa il push della parentesi incontrata
- Se si incontra una parentesi chiusa si fa il pop e si controlla che la parentesi estratta dallo stack sia dello stesso tipo di quella incontrata

Esercizio

- Scrivere una funzione che riceve in input una stringa contenente parentesi tonde e restituisce true se le parentesi sono ben accoppiate e false altrimenti.
  - Esempio:
    - x((abc()) è un'espressione ben parentesizzata
    - (as(df)))(non è un'espressione ben parentesizzata
- Modificare la funzione in modo che funzioni con stringhe contenenti sia parentesi tonde che parentesi quadrate
  - Esempio:
    - x[(abc)()]([]) è un'espressione ben parentesizzata
    - (as(df)](non è un'espressione ben parentesizzata