Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

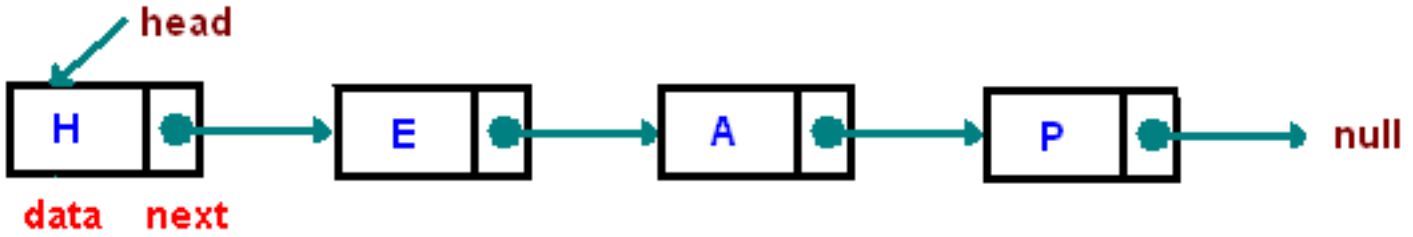
Università degli Studi dell'Aquila

Corso di Algoritmi e Strutture Dati con Laboratorio

La classe `SinglyLinkedList<E>`

SinglyLinkedList: una classe “giocattolo”

- Una linked list è una struttura dati dinamica lineare in cui ogni elemento (detto **nodo**) è un oggetto separato



- Ogni nodo è composto da due items: il dato ed un riferimento al nodo successivo
- L’ultimo nodo ha un riferimento a `null`.
- Il (riferimento al) primo nodo è detto “**`head`**” della lista. Se la lista è vuota `head` è un riferimento `null`.

Inner classes (cenni)

- The Java programming language allows you to define a class (B) within another class (A)
- Such a class is called a *nested / inner class* (**classe annidata / interna**) and is illustrated here:

```
class OuterClass {
    ... class InnerClass { ... }
}
```

Inner classes are divided into two categories:

- **static inner classes.**
- non-static inner classes, simply called **inner classes.**

Inner classes (cenni)

```
class OuterClass {
     ...
     static class StaticInnerClass { ... }
     ...
     class InnerClass { ... }
}
```

- A inner class is a member of its enclosing class
- Non-static **inner classes** have access to other members of the enclosing class, even if they are declared private, and vice versa.
- **Static inner classes** do not have access to other members of the enclosing class. Static inner classes have only access to static members of the enclosing class.
- As a member of the OuterClass, a nested class can be declared private, public, protected, or *package private*. (Recall that outer classes can only be declared public or *package private*)

Inner classes (cenni)

- Un esempio di accesso ad un membro della classe esterna:

```
class ClasseEsterna {
  private int x = 9;

  class ClasseInterna {
      public void stampaX() {
        System.out.println("Il valore di x è " + x);
      }
  }//end-ClasseInterna

}//end-ClasseEsterna
```

- Il codice precedente stamperà il numero 9 poiché l'Inner Class è in grado di accedere ai membri dell'Outer Class in maniera diretta anche se dichiarati privati.

Linked Lists: la classe `Node`

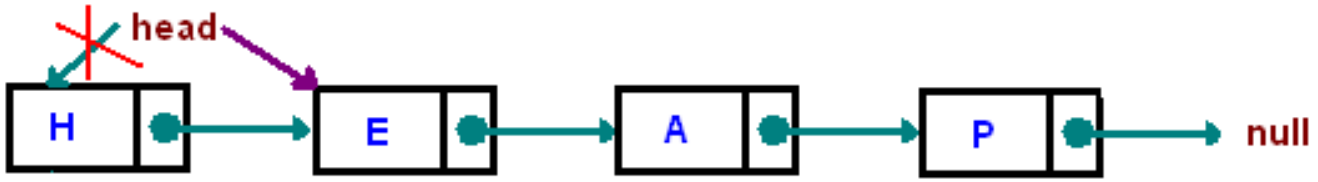
- In seguito definiamo una classe `SinglyLinkedList` con due inner classes: **`static Node class`** and non-static **`LinkedListIterator class`**

[rif. SinglyLinkedList.java]

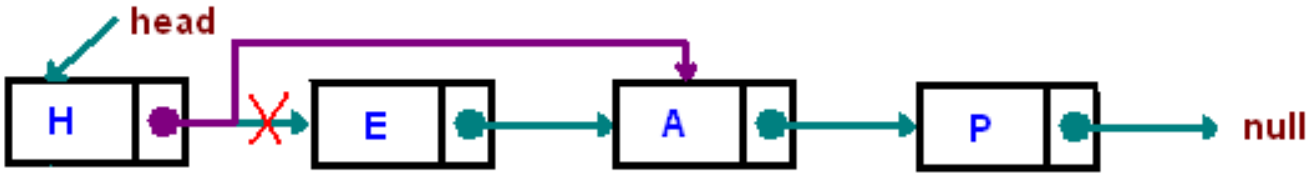
```
private static class Node<AnyType> {
      private AnyType data;
      private Node<AnyType> next;

      public Node(AnyType data, Node<AnyType>
next){
            this.data = data;
            this.next = next;
      }
}
```

Esempi

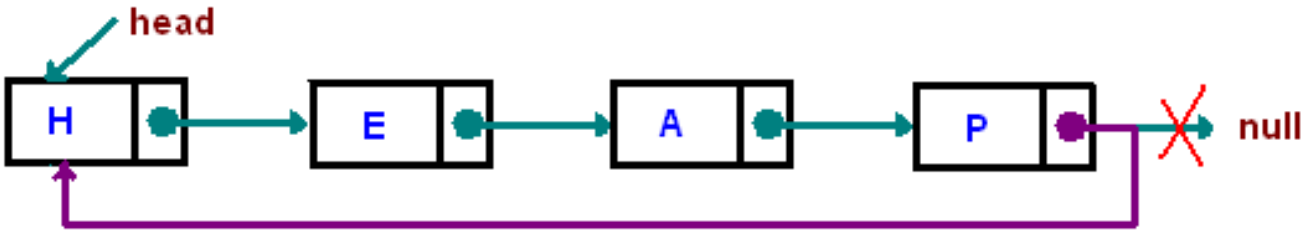
```
head = head.next;
```




```
head.next = head.next.next;
```



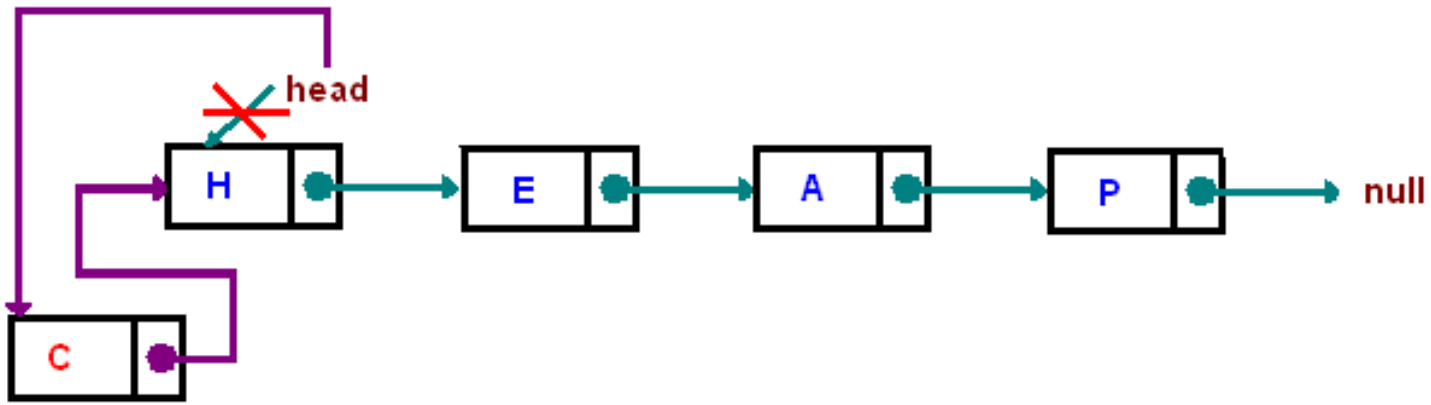

```
head.next.next.next.next = head;
```



Linked List Operations

- **addFirst** crea un nodo e lo aggiunge all'inizio della lista:

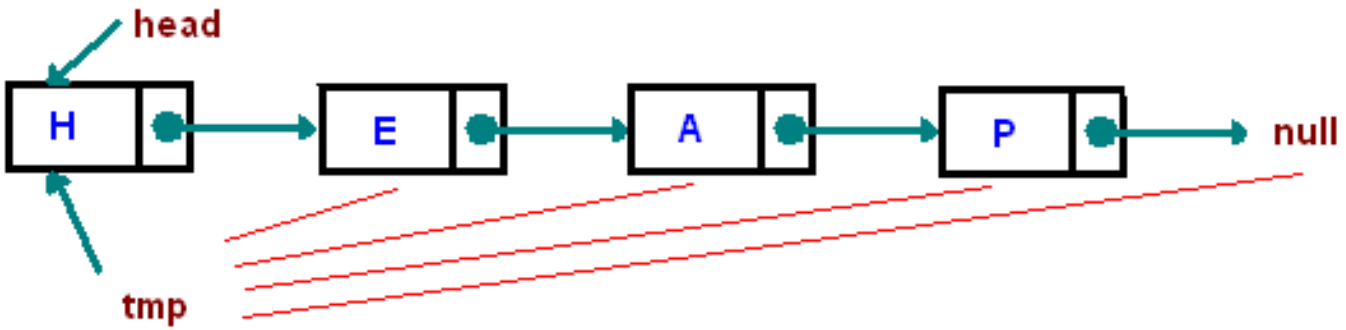
```
public void addFirst(AnyType item) {
  head = new Node<AnyType>(item, head);
}
```



Linked List Operations

- **Traversing** Iniziando da `head` (senza cambiare il riferimento) si accede ad ogni nodo fino a quando non si raggiunge `null`.

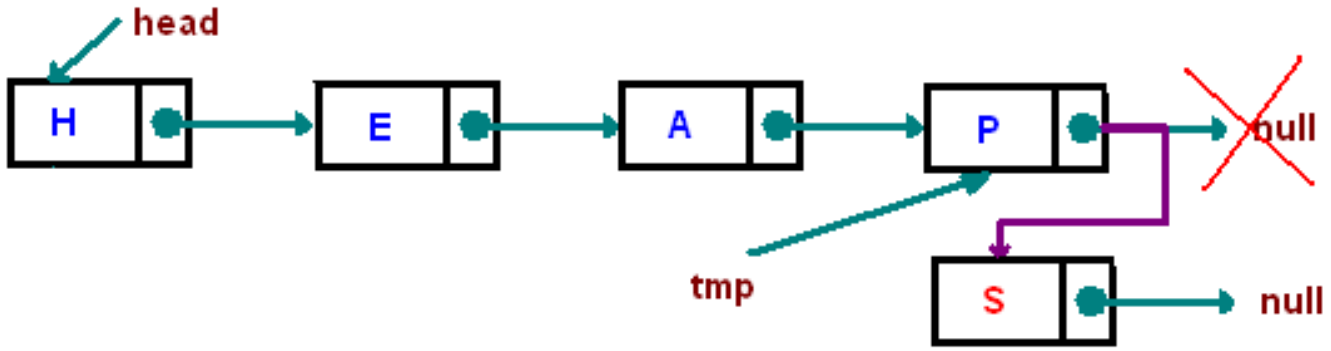
```
Node tmp = head;
while(tmp != null) tmp = tmp.next;
```



Linked List Operations

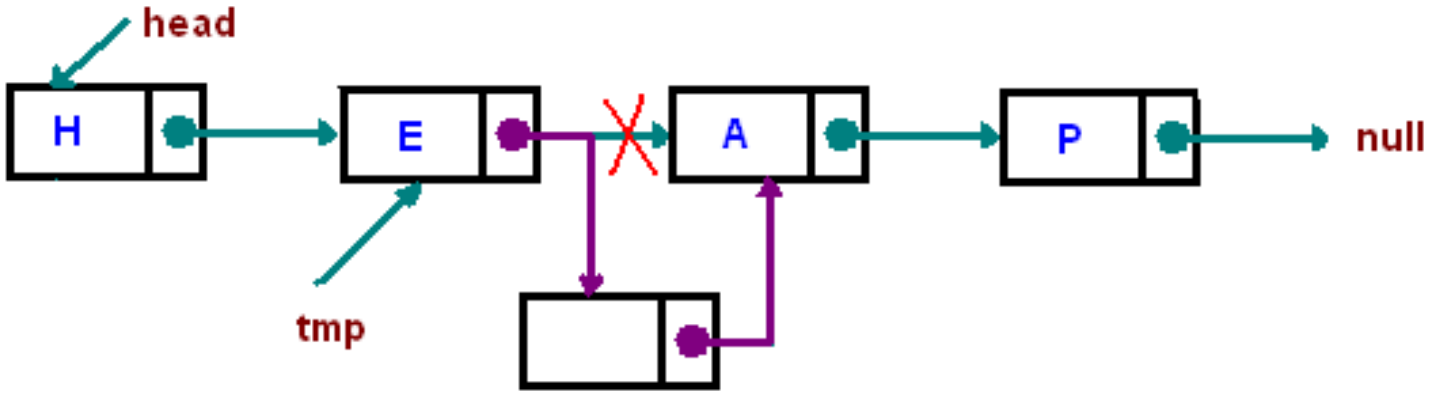
- **addLast** appende il nodo alla fine della lista.

```
public void addLast(AnyType item) {
   if(head == null) addFirst(item);
   else {
      Node<AnyType> tmp = head;
      while(tmp.next != null) tmp = tmp.next;
      tmp.next = new Node<AnyType>(item, null);
   }
}
```



Linked List Operations

- **Inserting "after“ (E)** : Cerca un nodo contenente “key” ed inserisce un nuovo nodo dopo di esso.

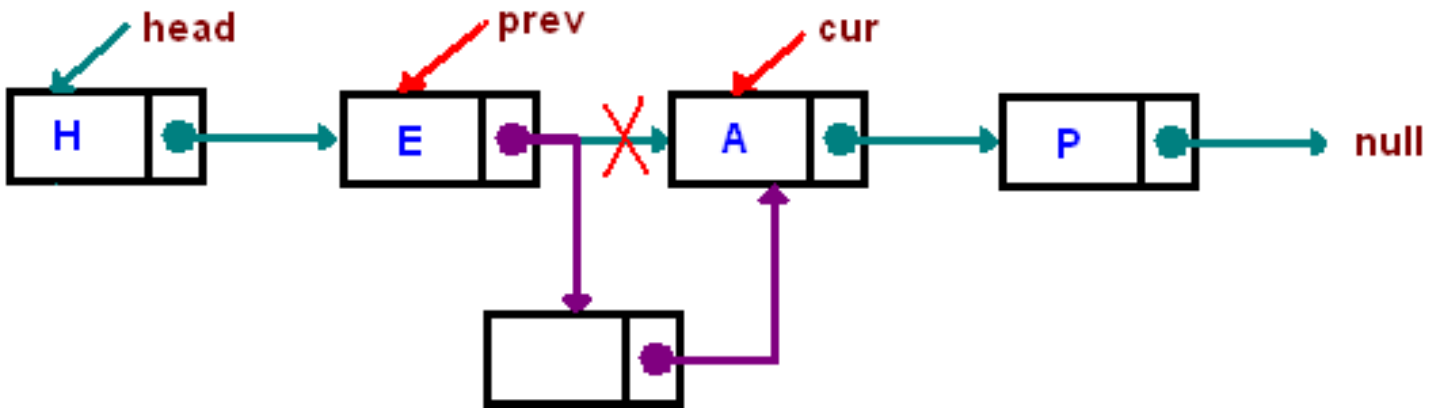


Linked List Operations

```
public void insertAfter(AnyType key, AnyType toInsert){
Node<AnyType> tmp = head;
while(tmp!=null && !tmp.data.equals(key))
      tmp = tmp.next;
if(tmp!=null) tmp.next =
      new Node<AnyType>(toInsert,tmp.next);
}
```

Linked List Operations

- **Inserting "before“ (A)** cerca un nodo contenente “key” e inserisce un nuovo nodo prima di esso



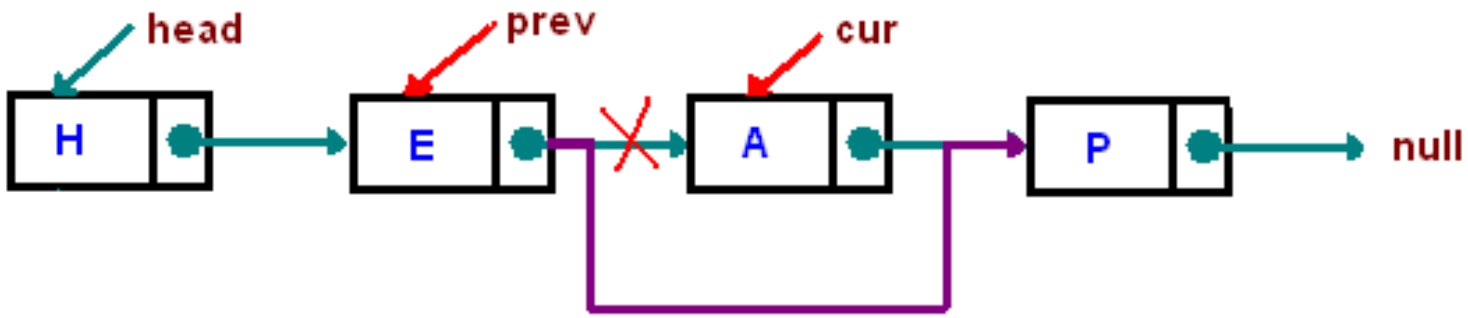
Linked List Operations

```
public void insertBefore(AnyType key,
                                 AnyType toInsert) {
  if(head == null) return null;
  if(head.data.equals(key)) {
     addFirst(toInsert);
     return; }
  Node<AnyType> prev = null;
  Node<AnyType> cur = head;
  while(cur != null && !cur.data.equals(key)) {
     prev = cur; cur = cur.next;
  }
  if(cur != null) prev.next =
        new Node<AnyType>(toInsert, cur);
}
```

Linked List Operations

- **Deletion (A)** Cerca un nodo contenente “key” e lo cancella. Ci sono tre casi particolari da gestire:

1. La lista è vuota
2. Bisogna cancellare il primo nodo
3. Il nodo non è in lista



Linked List Operations

```
public void remove(AnyType key) {
    if(head == null)
       throw new RuntimeException("cannot delete");
    if( head.data.equals(key) ) {
       head = head.next;
       return; }
    Node<AnyType> cur = head;
    Node<AnyType> prev = null;
    while(cur != null && !cur.data.equals(key) ) {
       prev = cur; cur = cur.next; }
    if(cur == null)
       throw new RuntimeException("cannot delete");
       prev.next = cur.next;
}
```

Linked List Operations

- **Iterator**

```
public Iterator<AnyType> iterator() {
   return new LinkedListIterator();
}
```

- **`LinkedListIterator`** è una classe privata interna alla classe `LinkedList`