Stackar

Koffman & Wolfgang

- kapitel 3

Stackar

Methods	Behavior
`boolean empty()`	Returns **`true`** if the stack is empty; otherwise, returns **`false`**.
`E peek()`	Returns the object at the top of the stack without removing it.
`E pop()`	Returns the object at the top of the stack and removes it.
`E push(E obj)`	Pushes an item onto the top of the stack and returns the item pushed.

Det ”översta” elementet är det senaste

- LIFO = Last-In-First-Out

Stackar används överallt i datorer

- t.ex. vid funktions-/metodanrop (run-time stack, call stack)

Balanserade parenteser

Vi behöver en stack för att kontrollera ifall ett uttryck har balanserade parenteser:

- speciellt om uttrycket kan innehålla olika sorters parenteser
- `(a+b*[c/{d-e}])+{d/e}`

Algoritmidé:

- när vi läser (, [eller { så lägger vi den på stacken
- när vi läser),] eller } så poppar vi stacken och kontrollerar att parenteserna stämmer

Stack som en utökning av Vector

java.util.Stack är en utökning av Vector (nästan som ArrayList):

```
public class Stack<E> extends Vector<E>
```

Topp-elementet är det sista elementet i vektorn

- push(e) implementeras med add(e)
- pop() implementeras med remove(size-1)

Detta är en ***stor miss*** i Java:

- eftersom Stack utökar Vector så får användaren använda alla Vector-metoder, såsom att läsa vilket element som helst
- Java har låst sig i sin implementation
- och om folk börjar utnyttja Vector-metoder så kan man få många oförklarliga fel

Implementera en stack

Vi kan implementera stacken som ett fält:

- vi måste kunna omallokera fältet
- det blir väldigt likt en ArrayList

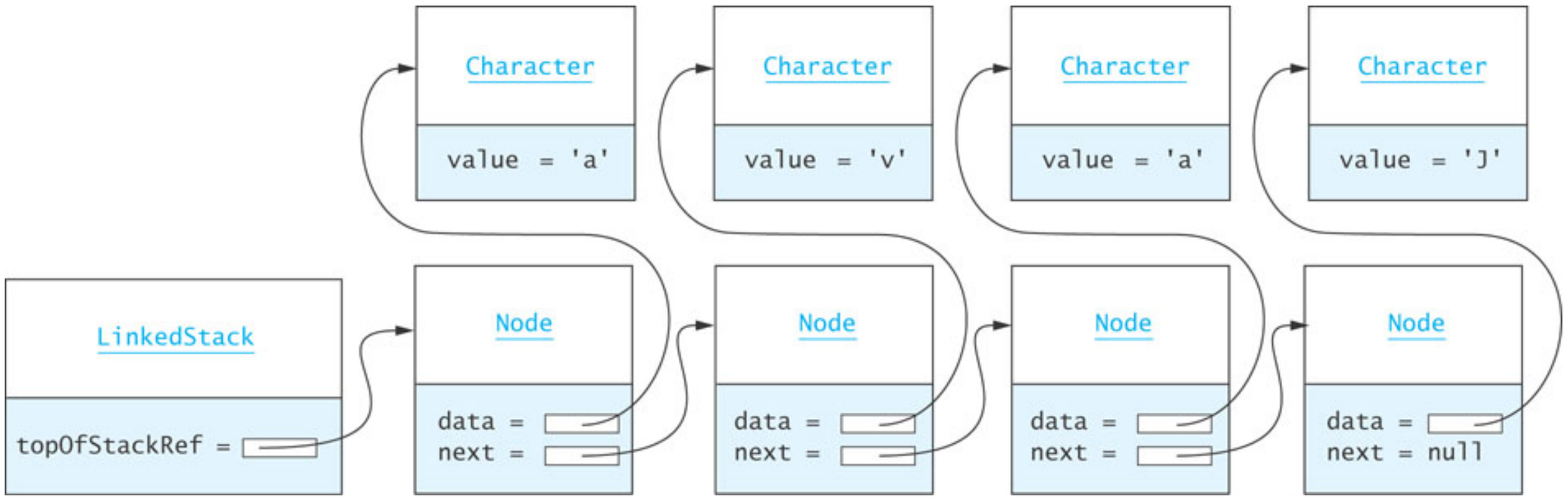
Vi kan implementera stacken som en länkad lista:

- det blir väldigt likt en LinkedList

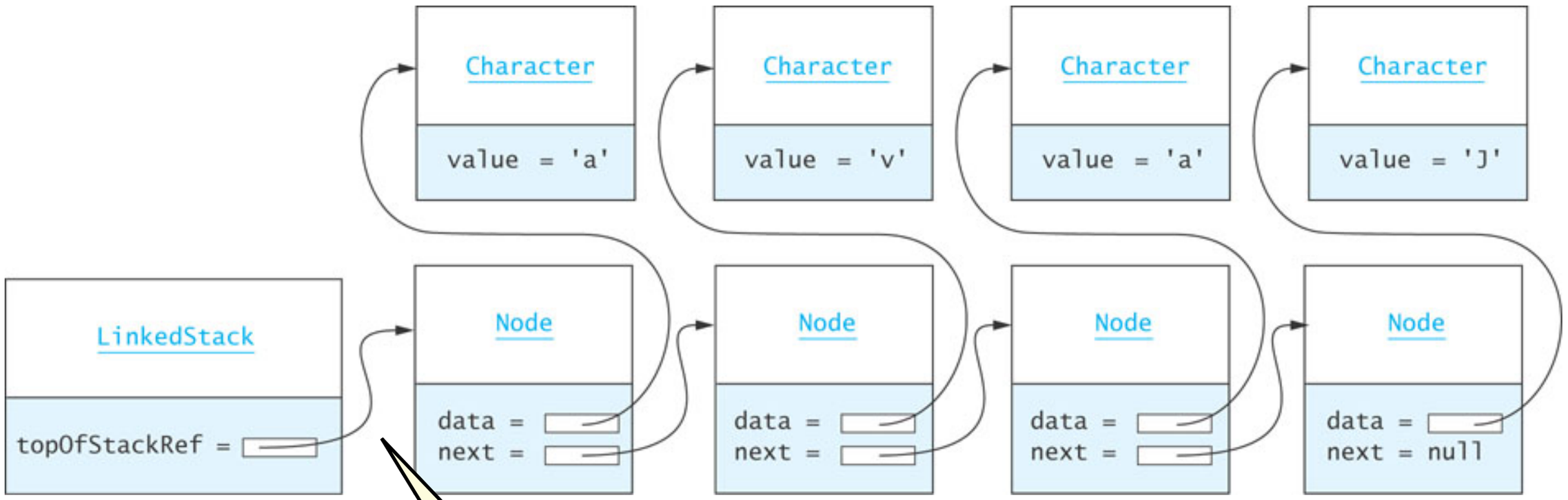
Vi kan också använda intern privat lista:

- t.ex. ArrayList eller LinkedList
- stackmetoderna (t.ex. `push(E)`) kommer då att anropa den interna listans metoder (t.ex. `add(E)`)
- det kallas *delegering*

En stack som en länkad lista



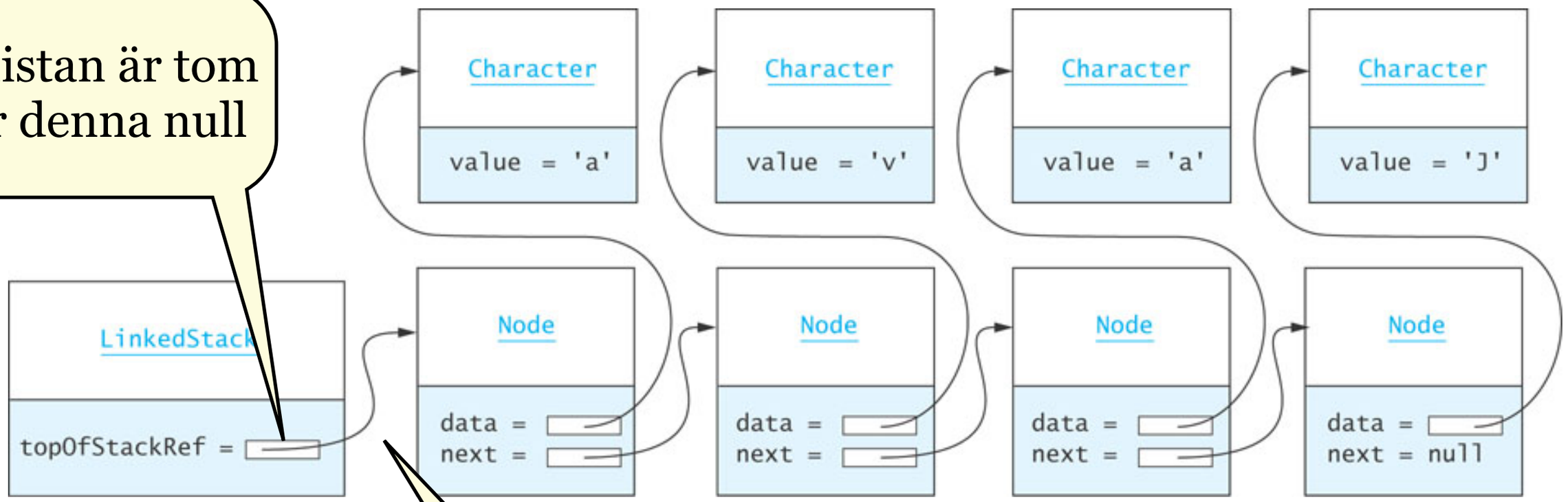
En stack som en länkad lista



Vi lägger till och tar bort element i början av listan

En stack som en länkad lista

När listan är tom så är denna null



Vi lägger till och tar bort element i början av listan

Komplexitet för stackmetoder

Den stora fördelen med stackar är att de är väldigt effektiva:

- alla operationer har konstant tidskomplexitet, O(1)
- dvs, push(E), pop(), peek() och empty()

Balanserade parenteser, iterativt

```
private static final String PARENS  = "()[]{}";

public static void parseParensIterative(String str) throws ParseException {
        Stack<Integer> parenStack = new Stack<Integer>();
        for (int i = 0; i < str.length(); i++) {
                char c = str.charAt(i);
                int paren = PARENS.indexOf(c);
                if (paren >= 0) {
                        if (paren % 2 == 0) {
                                parenStack.push(paren);
                        } else {
                                if (parenStack.empty())
                                        throw new ParseException("Too many close parentheses", i);
                                int openParen = parenStack.pop();
                                if (paren != openParen + 1)
                                        throw new ParseException("Unbalanced parentheses", i);
                        }
                }
        }
        if (!parenStack.empty())
                throw new ParseException("Too many open parentheses", str.length());
}
```

Balanserade parenteser, iterativt

```
private static final String PARENS  = "()[]{}";

public static void parseParensIterative(String str) throws ParseException {
        Stack<Integer> parenStack = new Stack<Integer>();
        for (int i = 0; i < str.length(); i++) {
                char c = str.charAt(i);
                int paren = PARENS.indexOf(c);
                if (paren >= 0) {
                        if (paren % 2 == 0) {
                                parenStack.push(paren);
                        } else {
                                if (parenStack.empty())
                                        throw new ParseException("Too many close parentheses", i);
                                int openParen = parenStack.pop();
                                if (paren != openParen + 1)
                                        throw new ParseException("Unbalanced parentheses", i);
                        }
                }
        }
        if (!parenStack.empty())
                throw new ParseException("Too many open parentheses", str.length());
}
```

vi behöver en stack för att hålla reda på parenteserna

Balanserade parenteser, iterativt

```
private static final String PARENS  = "()[]{}";

public static void parseParensIterative(String str) throws ParseException {
        Stack<Integer> parenStack = new Stack<Integer>();
        for (int i = 0; i < str.length(); i++) {
                char c = str.charAt(i);
                int paren = PARENS.indexOf(c);
                if (paren >= 0) {
                        if (paren % 2 == 0) {
                                parenStack.push(paren);
                        } else {
                                if (parenStack.empty())
                                        throw new ParseException("Too many close parentheses", i);
                                int openParen = parenStack.pop();
                                if (paren != openParen + 1)
                                        throw new ParseException("Unbalanced parentheses", i);
                        }
                }
        }
        if (!parenStack.empty())
                throw new ParseException("Too many open parentheses", str.length());
}
```

vi behöver en stack för att hålla reda på parenteserna

detta förutsätter att ([{ ligger på jämna index i PARENS

Balanserade parenteser, iterativt

```
private static final String PARENS  = "()[]{}";

public static void parseParensIterative(String str) throws ParseException {
        Stack<Integer> parenStack = new Stack<Integer>();
        for (int i = 0; i < str.length(); i++) {
                char c = str.charAt(i);
                int paren = PARENS.indexOf(c);
                if (paren >= 0) {
                        if (paren % 2 == 0) {
                                parenStack.push(paren);
                        } else {
                                if (parenStack.empty())
                                        throw new ParseException("Too many close parentheses", i);
                                int openParen = parenStack.pop();
                                if (paren != openParen + 1)
                                        throw new ParseException("Unbalanced parentheses", i);
                        }
                }
        }
        if (!parenStack.empty())
                throw new ParseException("Too many open parentheses", str.length());
}
```

vi behöver en stack för att hålla reda på parenteserna

detta förutsätter att ([{ ligger på jämna index i PARENS

detta förutsätter att (och) ligger efter varandra i PARENS