Stacks are LIFO

- A stack
  - Last-in, first-out (LIFO) property
    - The last item placed on the stack will be the first item removed
  - Analogy
    - A stack of dishes in a cafeteria

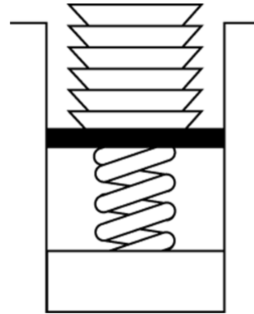


Figure 7-1
Stack of cafeteria dishes

© 2006 Pearson Addison-Wesley. All rights reserved

A Stack ADT

- ADT stack operations
  - Create an empty stack
  - Determine whether a stack is empty
  - Add a new item to the stack
  - Remove from the stack the item that was added most recently
  - Remove all the items from the stack
  - Retrieve from the stack the item that was added most recently

© 2006 Pearson Addison-Wesley. All rights reserved

Refining the Definition of the ADT Stack

- Pseudocode for the ADT stack operations

```
createStack()
// Creates an empty stack.

isEmpty()
// Determines whether a stack is empty.

push(newItem) throws StackException
// Adds newItem to the top of the stack.
// Throws StackException if the insertion is
// not successful.
```

© 2006 Pearson Addison-Wesley. All rights reserved

Refining the Definition of the ADT Stack

- Pseudocode for the ADT stack operations (Continued)

```
pop() throws StackException
// Retrieves and then removes the top of the stack.
// Throws StackException if the deletion is not
// successful.

popAll()
// Removes all items from the stack.

peek() throws StackException
// Retrieves the top of the stack. Throws
// StackException if the retrieval is not successful
```

© 2006 Pearson Addison-Wesley. All rights reserved

Simple Applications of the ADT Stack: Checking for Balanced Braces

- A stack can be used to verify whether a program contains balanced braces (but there is a much simpler way)
  - An example of balanced braces

    `abc{defg{ijk}{l{mn}}op}qr`
  - An example of unbalanced braces

    `abc{def}}{ghij{kl}m`

© 2006 Pearson Addison-Wesley. All rights reserved

Checking for Balanced Braces

- Requirements for balanced braces
  - Each time you encounter a "}", it matches an already encountered "{"
  - When you reach the end of the string, you have matched each "{"

© 2006 Pearson Addison-Wesley. All rights reserved

Checking for Balanced Braces

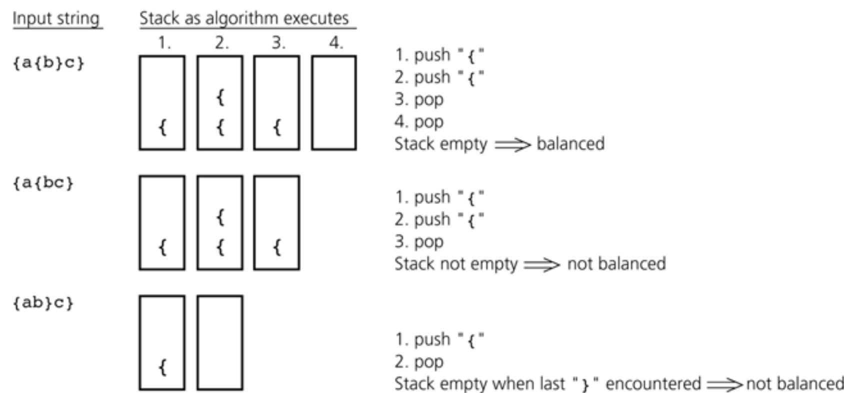


Figure 7-3

Traces of the algorithm that checks for balanced braces

© 2006 Pearson Addison-Wesley. All rights reserved

Recognizing Strings in a Language

- Language L

  L = { w$w' : w is a possible empty string of characters other than $, w ' = reverse(w) }
  - A stack can be used to determine whether a given string is in L
    - Traverse the first half of the string, pushing each character onto a stack
    - Once you reach the $, for each character in the second half of the string, pop a character off the stack
      - Match the popped character with the current character in the string

© 2006 Pearson Addison-Wesley. All rights reserved

Implementations of the ADT Stack

- The ADT stack can be implemented using
  - An array
  - A linked list
  - The ADT list
- `StackInterface`
  - Provides a common specification for the three implementations
- `StackException`
  - Used by `StackInterface`
  - Extends `java.lang.RuntimeException`

© 2006 Pearson Addison-Wesley. All rights reserved

Implementations of the ADT Stack

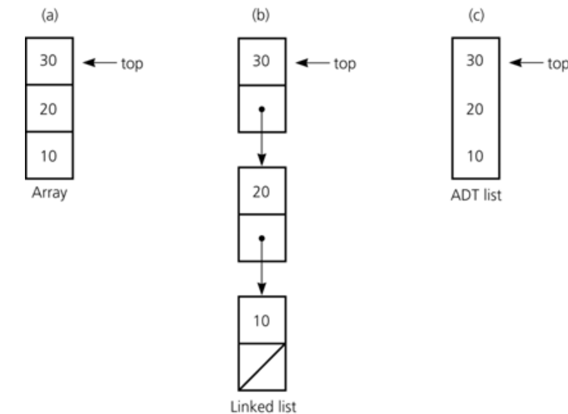


Figure 7-4

Implementation of the ADT stack that use a) an array; b) a linked list; c) an ADT list

© 2006 Pearson Addison-Wesley. All rights reserved

An Array-Based Implementation of the ADT Stack

- `StackArrayBased` class
  - Implements `StackInterface`
  - Instances
    - Stacks
  - Private data fields
    - An array of `Objects` called `items`
    - The index `top`

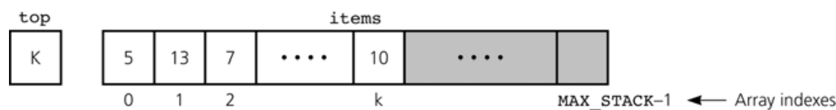


Figure 7-5

An array-based implementation

© 2006 Pearson Addison-Wesley. All rights reserved

An Array-Based Implementation of the ADT Stack

```
final int MAX_STACK = 50;  // maximum size of stack
private Object items[];
private int top;

public StackArrayBased() {
  items = new Object[MAX_STACK];
  top = -1;
}
public boolean isEmpty() {
  return top < 0;
}
public boolean isFull() {
  return top == MAX_STACK-1;
}
```

© 2006 Pearson Addison-Wesley. All rights reserved

An Array-Based Implementation of the ADT Stack

```
public void push(Object newItem) throws StackException {
  if (!isFull()) {
    items[++top] = newItem;
  }
  else {
    throw new StackException("StackException on " +
                             "push: stack full");
  }
}

public Object pop() throws StackException {
  if (!isEmpty()) {
    return items[top--];
  }
  else {
    throw new StackException("StackException on " +
                             "pop: stack empty");
  }
}
```

© 2006 Pearson Addison-Wesley. All rights reserved

A Reference-Based Implementation of the ADT Stack

- A reference-based implementation
  - Required when the stack needs to grow and shrink dynamically
- `StackReferenceBased`
  - Implements `StackInterface`
  - `top` is a reference to the head of a linked list of items

© 2006 Pearson Addison-Wesley. All rights reserved

A Reference-Based Implementation of the ADT Stack

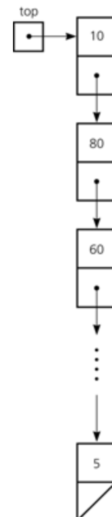


Figure 7-6

A reference-based implementation

© 2006 Pearson Addison-Wesley. All rights reserved

A Reference-Based Stack

```
public StackReferenceBased() {
  top = null;
}
public boolean isEmpty() {
  return top == null;
}
public void push(Object newItem) {
  top = new Node(newItem, top);
}
public Object pop() throws StackException {
  if (!isEmpty()) {
    Node temp = top;
    top = top.getNext();
    return temp.getItem();
  }
  else {
    throw new StackException("StackException on " +
                             "pop: stack empty");
  }
}
```

© 2006 Pearson Addison-Wesley. All rights reserved

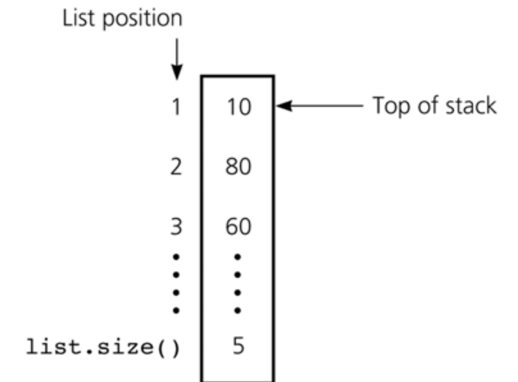
An Implementation That Uses the ADT List

- The ADT list can be used to represent the items in a stack
- If the item in position 1 of a list represents the top of the stack
  - `push(newItem)` operation is implemented as
    `add(1, newItem)`
  - `pop()` operation is implemented as
    `get(1)`
    `remove(1)`
  - `peek()` operation is implemented as
    `get(1)`

© 2006 Pearson Addison-Wesley. All rights reserved

An Implementation That Uses the ADT List

Figure 7-7
An implementation that uses the ADT list



© 2006 Pearson Addison-Wesley. All rights reserved

ADT List Implementation

```
public StackListBased() {
  list = new ListReferenceBased();
}
public boolean isEmpty() {
  return list.isEmpty();
}
public void push(Object newItem) {
  list.add(1, newItem);
}
public Object pop() throws StackException {
  if (!list.isEmpty()) {
    Object temp = list.get(1);
    list.remove(1);
    return temp;
  }
  else {
    throw new StackException("StackException on " +
                                        "pop: stack empty");
  }
}
```

© 2006 Pearson Addison-Wesley. All rights reserved