What is a Stack in Java?

A **Stack** in Java is a **Last-In, First-Out (LIFO)** data structure. The last element added to the stack is the first one to be removed. Java provides a built-in `Stack` class in the `java.util` package.

java
CopyEdit

```
Stack<Integer> stack = new Stack<>();
stack.push(10);
stack.push(20);
System.out.println(stack.pop()); // Outputs: 20
```

Inheritance: `Stack` extends `Vector`

The `Stack` class **inherits** from the `Vector` class, which means it inherits **all public and protected methods** of `Vector`.

✅ **Methods inherited from `Vector` include:**

- `add(E e)` – Adds an element at the end
- `add(int index, E element)` – Inserts element at the specified index
- `get(int index)` – Returns the element at the specified index
- `remove(int index)` – Removes element at the specified index
- `size()` – Returns the number of elements
- `clear()` – Removes all elements
- `contains(Object o)` – Checks if the element is present
- `isEmpty()` – Checks if the vector is empty
- `iterator()` – Returns an iterator over the elements

✅ **Stack-specific methods (LIFO methods):**

```
public E push(E item);     // Adds an item to the top
public synchronized E pop();   // Removes and returns the top item
```

```
public synchronized E peek();  // Looks at the top item without
removing
public boolean empty();        // Checks if the stack is empty
public int search(Object o);   // Returns the position from the top
```

Is Stack Used in Real Life?

Yes, **stacks are used frequently in real-world applications**, especially where LIFO behavior is needed:

✅ **Real-life Use Cases:**

1. **Undo/Redo Operations** – Text editors, photo editors.
2. **Browser History** – Back/forward navigation.
3. **Expression Evaluation** – In compilers or calculators.
4. **Recursive Algorithms** – When using iterative approach with explicit stack.
5. **Parsing and Tree Traversal** – HTML/XML parsers, compilers, tree algorithms.
6. **Language Runtime Call Stack** – Function call tracking.

Example: Undo Operation

java
CopyEdit

```
Stack<String> undoStack = new Stack<>();
undoStack.push("Type A");
undoStack.push("Type B");
System.out.println("Undo: " + undoStack.pop()); // Undo: Type B
```

Important Note

Although `Stack` is available, **it's considered outdated** in favor of **`Deque` implementations** like `ArrayDeque`, which are faster and more flexible:

```
Deque<Integer> stack = new ArrayDeque<>();
stack.push(1);   // push
```

```
stack.pop();      // pop
stack.peek();     // peek
```

Let me know if you'd like a comparison between `Stack` and `Deque` or real-world code examples.