

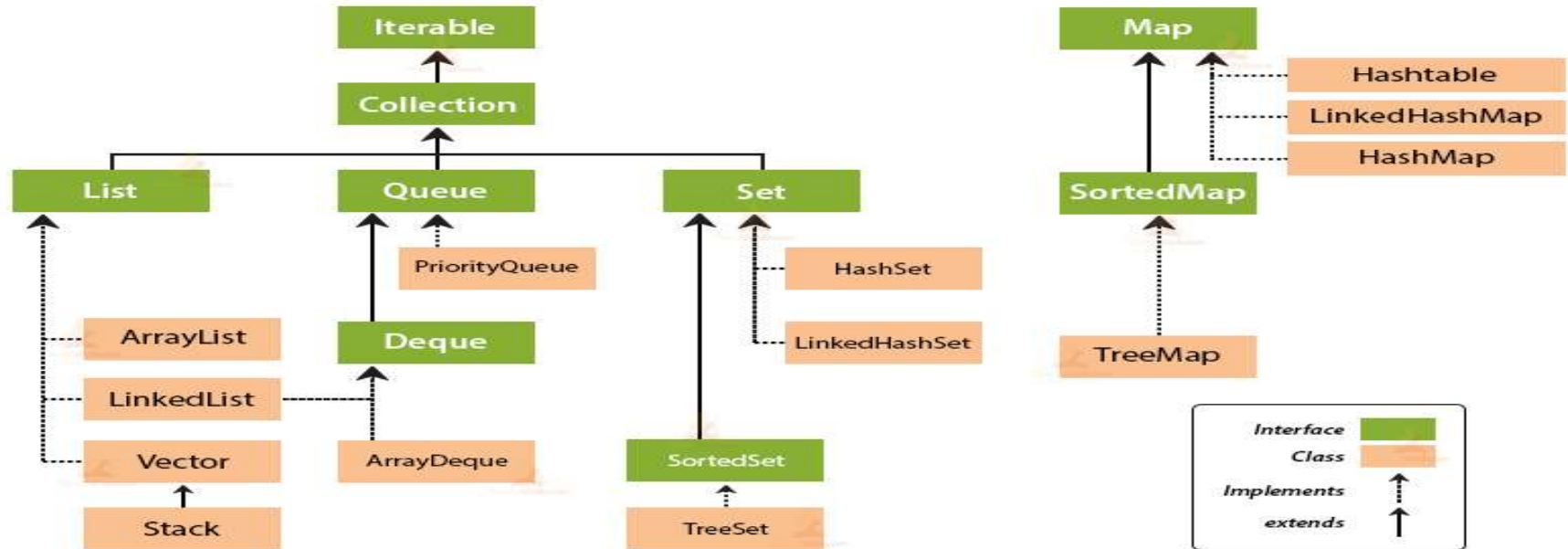
Stack and Queue

What you'll Learn

- Introduction of Stack & Queue
- Internal storage structure
- Methods and some coding examples
- Common operations of queue(insert,delete)



Collection Framework Hierarchy in Java



Stack

- It provides a Stack class which models and implements Stack data structure.
- A Stack is a data structure where you add elements to the "top" of the stack, and also remove elements from the top again (LIFO).
- In addition to the basic push and pop operations, the class provides three more functions of empty, search and peek.
- Also referred to as the subclass of Vector.
- The class supports one default constructor Stack() which is used to create an empty stack.

Internal Storage Structure

- The JVM divided the memory into following sections.
 - **Heap** - The Heap section contains Objects (may also contain reference variables).
 - **Stack** - The Stack section of memory contains methods, local variables, and reference variables.
 - **Code** - The code section contains your bytecode.
 - **Static** - The Static section contains Static data/methods.



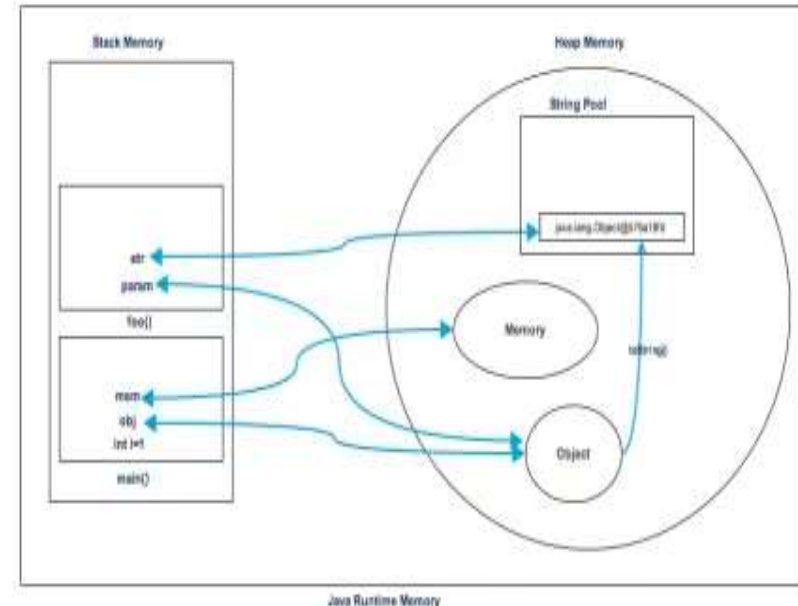
Internal Storage Structure

- Java Stack memory is used for the execution of a thread.
- They contain method-specific values that are short-lived and references to other objects in the heap that is getting referred from the method.
- Stack memory is always referenced in LIFO (Last-In-First-Out) order.
- Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method.
- As soon as the method ends, the block becomes unused and becomes available for the next method.



Memory Example

```
public class Main {
    public static void main(String[] args) {
        int i=1;
        Object obj = new Object();
        Main mem = new Main();
        mem.foo(obj);
    }
    private void foo(Object param) {
        String str = param.toString();
        System.out.println(str);
    }
}
```



Methods

Methods	Description
boolean empty()	Tests if this stack is empty. Returns true if the stack is empty, and returns false if the stack contains elements.
E peek()	Looks at the object at the top of this stack without removing it from the stack.
E pop()	Removes the object at the top of this stack and returns that object as the value of this function.
E push(Object element)	Pushes an item onto the top of this stack.
int search(Object element)	Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, -1 is returned.

Creating Stack

- To use a Java Stack you must first create an instance of the Stack class.
- Here is an example of creating a Java Stack instance:

```
Stack stack = new Stack();
```



Creating Stack

- You can set a generic type on a Stack specifying the type of objects the Stack instance can contain.
- You specify the stack type when you declare the Stack variable.
- Here is an example of creating a Java Stack with a generic type:
 - `Stack<String> stack = new Stack<String>();`
- The Stack created above can only contain String instances.



push() Element

- Once you have a Java Stack instance, you can push elements to the top of the Stack.
- You push elements onto a Java Stack using its push() method.
- Here is an example of pushing an element (object) onto a Java Stack:
 - `Stack<String> stack = new Stack<String>();`
 - `stack.push("1");`
- This Java example pushes a [Java String](#) with the text 1 onto the Stack.
- The String 1 is then stored at the top of the Stack.



Pop() Element

- Once an element has been pushed onto a Java Stack, you can pop that element from the Stack again.
- You pop an element off a Java Stack using the pop() method.
- Here is an example of popping an element off a Stack using the pop() method:
 - `Stack<String> stack = new Stack<String>();`
 - `stack.push("1");`
 - `String topElement = stack.pop();`
- Once an element is popped off a Stack, the element is no longer present on the Stack.

peek () Element

- The Java Stack class has a method called peek() which enables you to see what the top element on the Stack is, without popping off the element.
- Here is an example of peeking at the top of a Java Stack:
 - `Stack<String> stack = new Stack<String>();`
 - `stack.push("1");`
 - `String topElement = stack.peek();`
- After running this Java example the topElement variable will contain the String object 1 which was pushed onto the Stack just before peek() was called.

Search() Element

- You can search for an object on the stack to get its index, using the search() method.
- Here is how you search a Stack for an object:

```
Stack<String> stack = new Stack<String>();  
stack.push("1");  
stack.push("2");  
stack.push("3")  
int index = stack.search("3"); //index = 1
```



Example

```
import java.util.Stack;
public class Main {
    public static void main(String a[]){
        Stack<Integer> stack = new Stack<>();
        System.out.println("Empty stack : " + stack);
        System.out.println("Empty stack : " + stack.isEmpty());
        stack.push(1001);
        stack.push(1002);
        stack.push(1003);
        stack.push(1004);
        System.out.println("Non-Empty stack : " + stack);
        System.out.println("Non-Empty stack: Pop Operation : " + stack.pop());
        System.out.println("Non-Empty stack : After Pop Operation : " + stack);
        System.out.println("Non-Empty stack : search() Operation : " +
stack.search(1002));
        System.out.println("Non-Empty stack : " + stack.isEmpty());
    }
}
```

Queue

- The Queue interface is available in `java.util` package and extends the `Collection` interface.
- The queue collection is used to hold the elements about to be processed and provides various operations like the insertion, removal etc.
- The Queue is used to insert elements at the end of the queue and removes from the beginning of the queue. It follows FIFO concept.
- All Queues except the Deques support insertion and removal at the tail and head of the queue respectively. The Deques support element insertion and removal at both ends.

60	70	200	250	300
----	----	-----	-----	-----

Methods

Methods	Description
add()	This method is used to add elements at the tail of queue.
peek()	This method is used to view the head of queue without removing it. It returns Null if the queue is empty.
element()	This method is similar to peek(). It throws NoSuchElementException when the queue is empty.
remove()	This method removes and returns the head of the queue. It throws NoSuchElementException when the queue is empty.
poll()	This method removes and returns the head of the queue. It returns null if the queue is empty.
size()	This method return the no. of elements in the queue.

Blocking Queues

- Blocking Queues do not accept null elements. If we perform any null related operation, it throws `NullPointerException`.
- Blocking Queues are used to implement Producer/Consumer based applications.
- Blocking Queues are thread-safe.



offer() Methods

- The offer() operation is used to insert new element into the queue.
- If it performs insert operation successfully, it returns “true” value. Otherwise it returns “false” value.
- Let us develop one simple example to demonstrate this functionality.




Example

```
import java.util.concurrent.*;

public class QueueOfferOperation {

    public static void main(String[] args) {
        BlockingQueue<String> queue = new ArrayBlockingQueue<>(2);
        System.out.println(queue.offer("one"));
        System.out.println(queue.offer("two"));
        System.out.println(queue);
        System.out.println(queue.offer("three"));
        System.out.println(queue);
    }
}
```




remove() Method

- The remove() operation is used to delete an element from the head of the queue.
- If it performs delete operation successfully, it returns the head element of the queue. Otherwise it throws `java.util.NoSuchElementException`.
- Let us develop one simple example to demonstrate this functionality.



Example

```
import java.util.*;
public class Main{
    public static void main(String[] args){
        Queue<String> queue = new LinkedList<>();
        queue.offer("one");
        queue.offer("two");
        System.out.println(queue);
        System.out.println(queue.remove());
        System.out.println(queue.remove());
        System.out.println(queue.remove());
    }
}
```



poll() Method

- The poll() operation is used to delete an element from the head of the queue.
- If it performs delete operation successfully, it returns the head element of the queue. Otherwise it returns "null" value.
- Let us develop one simple example to demonstrate this functionality.



Example

```
import java.util.*;

public class Main{

    public static void main(String[] args){
        Queue<String> queue = new LinkedList<>();
        queue.offer("one");
        queue.offer("two");
        System.out.println(queue);
        System.out.println(queue.poll());
        System.out.println(queue.poll());
        System.out.println(queue.poll());
    }
}
```



peek() Methods

- The peek() operation is used to retrieve an element from the head of the queue, without removing it.
- If it performs examine operation successfully, it returns the head element of the queue. Otherwise it returns null value.
- Let us develop one simple example to demonstrate this functionality.



peek() - Example

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();
        queue.add("one");
        System.out.println(queue.peek());
        System.out.println(queue);
        queue.clear();
        System.out.println(queue.peek());
    }
}
```



Example

```
import java.util.LinkedList;
import java.util.Queue;
public class Main{
    public static void main(String[] args){
        Queue<Integer> q = new LinkedList<>();
        for (int i=0; i<5; i++)
            q.add(i);
        System.out.println("Elements of queue-"+q);
        int removedele = q.remove();
        System.out.println("removed element-" + removedele);
        System.out.println(q);
        int head = q.peek();
        System.out.println("head of queue-" + head);
        int size = q.size();
        System.out.println("Size of queue-" + size);
    }
}
```

Technical Questions

1. What is Data Structure? Explain.
2. Describe the types of Data Structures?
3. List the area of applications of Data Structure.
4. Which data structure is used to perform recursion?
5. What is a Stack?
6. List the area of applications where stack data structure can be used?
7. What are the operations that can be performed on a stack?
8. Write the stack overflow condition?
9. What is the difference between PUSH and POP?
10. Define the queue data structure?
11. List some applications of queue data structure?

Thank You!

