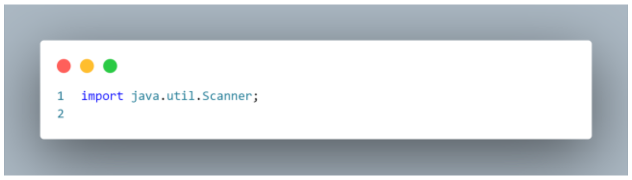
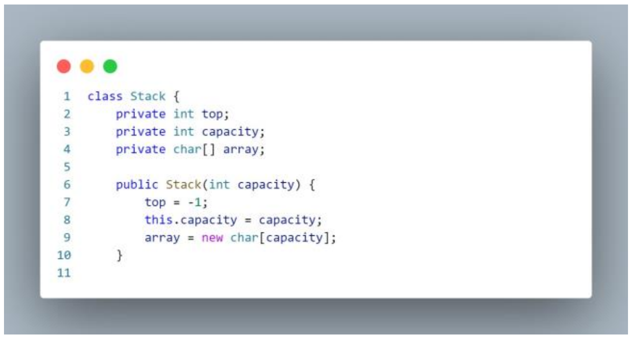
Balanced_Parenthesis_Check



```java
import java.util.Scanner;

```

Stack Class



```java
class Stack {
    private int top;
    private int capacity;
    private char[] array;

    public Stack(int capacity) {
        top = -1;
        this.capacity = capacity;
        array = new char[capacity];
    }

```

This class represents a basic stack data structure using an array.

Constructor `Stack(int capacity)`

- **Purpose**: Initializes the stack with a specified capacity.

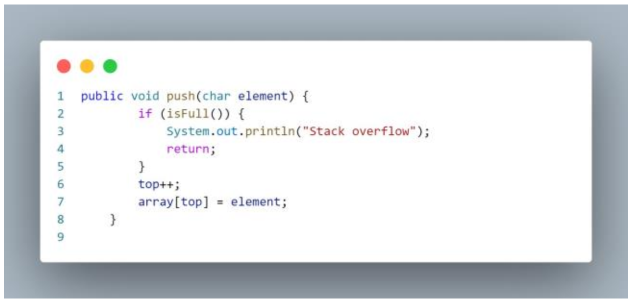
- **Parameters**: `capacity` - an integer indicating the size of the stack.

- **Behavior**:

- Sets `top` to -1 to indicate an empty stack.

- Initializes the `array` with the given `capacity`.

`push(char element)`



```java
public void push(char element) {
        if (isFull()) {
            System.out.println("Stack overflow");
            return;
        }
        top++;
        array[top] = element;
    }

```

- **Purpose**: Pushes an element onto the stack.

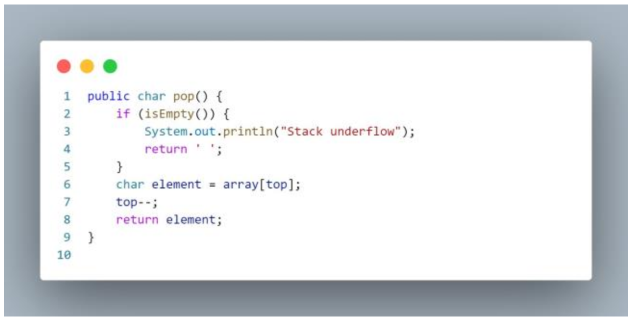
- **Parameters**: `element` - the character element to be pushed onto the stack.

- **Behavior**:

- Checks if the stack is already full (`isFull()` method).

- If not full, increments `top` and adds the `element` to the `array`.

`pop()`



```java
public char pop() {
    if (isEmpty()) {
        System.out.println("Stack underflow");
        return ' ';
    }
    char element = array[top];
    top--;
    return element;
}

```

- **Purpose**: Pops/removes the top element from the stack.

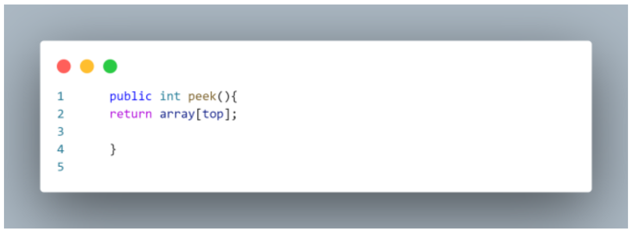
- **Returns**: The character element popped from the stack or a space character if the stack is empty.

- **Behavior**:

- Checks if the stack is empty (`isEmpty()` method).

- If not empty, retrieves the top element, decrements `top`, and returns the element.

`peek()`



```java
    public int peek(){
    return array[top];

    }

```

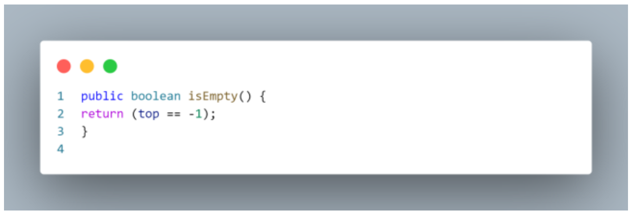
- **Purpose**: Returns the element at the top of the stack without removing it.

- **Returns**: The character element at the top of the stack.

- **Behavior**:

- Returns the element at the top of the stack (`array[top]`).

`isEmpty()`



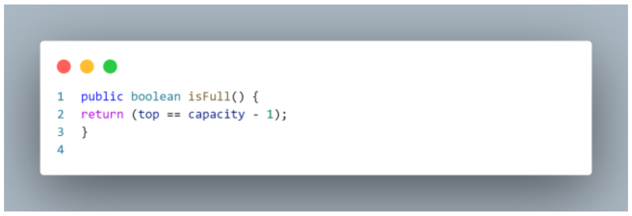
```java
public boolean isEmpty() {
return (top == -1);
}

```

- **Purpose**: Checks if the stack is empty.

- **Returns**: `true` if the stack is empty, `false` otherwise.

`isFull()`



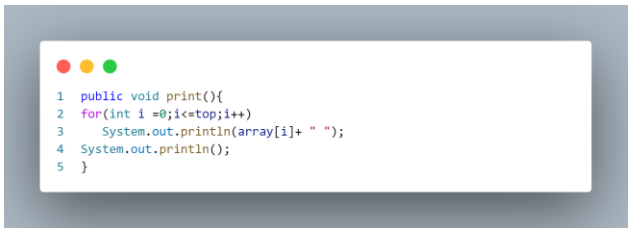
```java
public boolean isFull() {
return (top == capacity - 1);
}

```

- - **Purpose**: Checks if the stack is full.

- **Returns**: `true` if the stack is full, `false` otherwise.

`print()`



```java
public void print(){
for(int i =0;i<=top;i++)
   System.out.println(array[i]+ " ");
System.out.println();
}
```

- **Purpose**: Prints all elements in the stack.

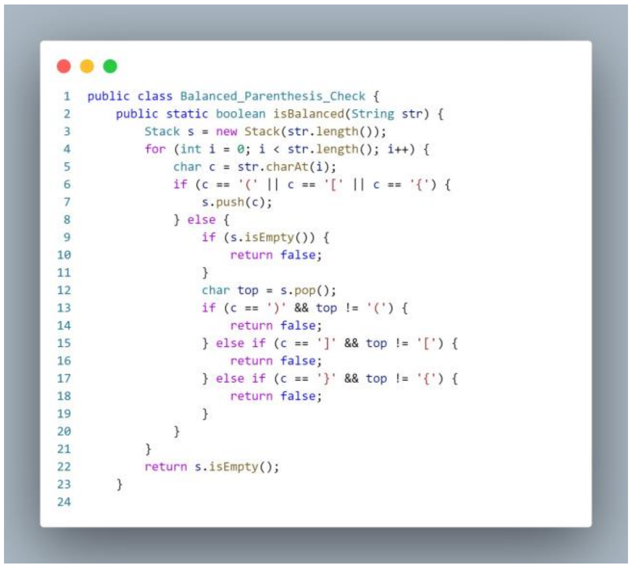
- **Behavior**:

- Loops through the elements in the stack and prints each element.

Balanced_Parenthesis_Check Class

This class contains methods for checking balanced parentheses.

`isBalanced(String str)`



```java
public class Balanced_Parenthesis_Check {
    public static boolean isBalanced(String str) {
        Stack s = new Stack(str.length());
        for (int i = 0; i < str.length(); i++) {
            char c = str.charAt(i);
            if (c == '(' || c == '[' || c == '{') {
                s.push(c);
            } else {
                if (s.isEmpty()) {
                    return false;
                }
                char top = s.pop();
                if (c == ')' && top != '(') {
                    return false;
                } else if (c == ']' && top != '[') {
                    return false;
                } else if (c == '}' && top != '{') {
                    return false;
                }
            }
        }
        return s.isEmpty();
    }

```

- **Purpose**: Checks if the parentheses in a string are balanced.

- **Parameters**: `str` - the input string containing parentheses.

- **Returns**: `true` if the parentheses are balanced, `false` otherwise.

- **Behavior**:

- Creates an instance of the `Stack` class.

- Iterates through each character in the input string.

- If the character is an opening parenthesis, pushes it onto the stack.

- If the character is a closing parenthesis:

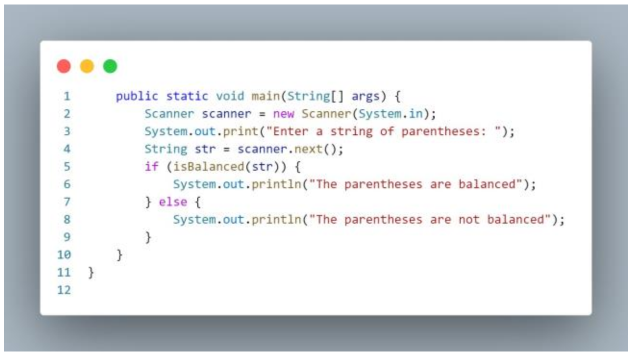
- Checks if the stack is empty. If so, returns `false` (unbalanced).

- Compares the closing parenthesis with the top element of the stack.

- If they don't match, returns `false`.

- Finally, checks if the stack is empty (all opening parentheses matched) and returns the result.

`main(String[] args)`



```java
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string of parentheses: ");
        String str = scanner.next();
        if (isBalanced(str)) {
            System.out.println("The parentheses are balanced");
        } else {
            System.out.println("The parentheses are not balanced");
        }
    }
}
```

- **Purpose**: Entry point of the program.

- **Behavior**:

- Takes user input for a string containing parentheses.

- Calls `isBalanced()` to check if the input string has balanced parentheses.

- Prints the result indicating whether the parentheses are balanced or not.

Hope this breakdown helps!