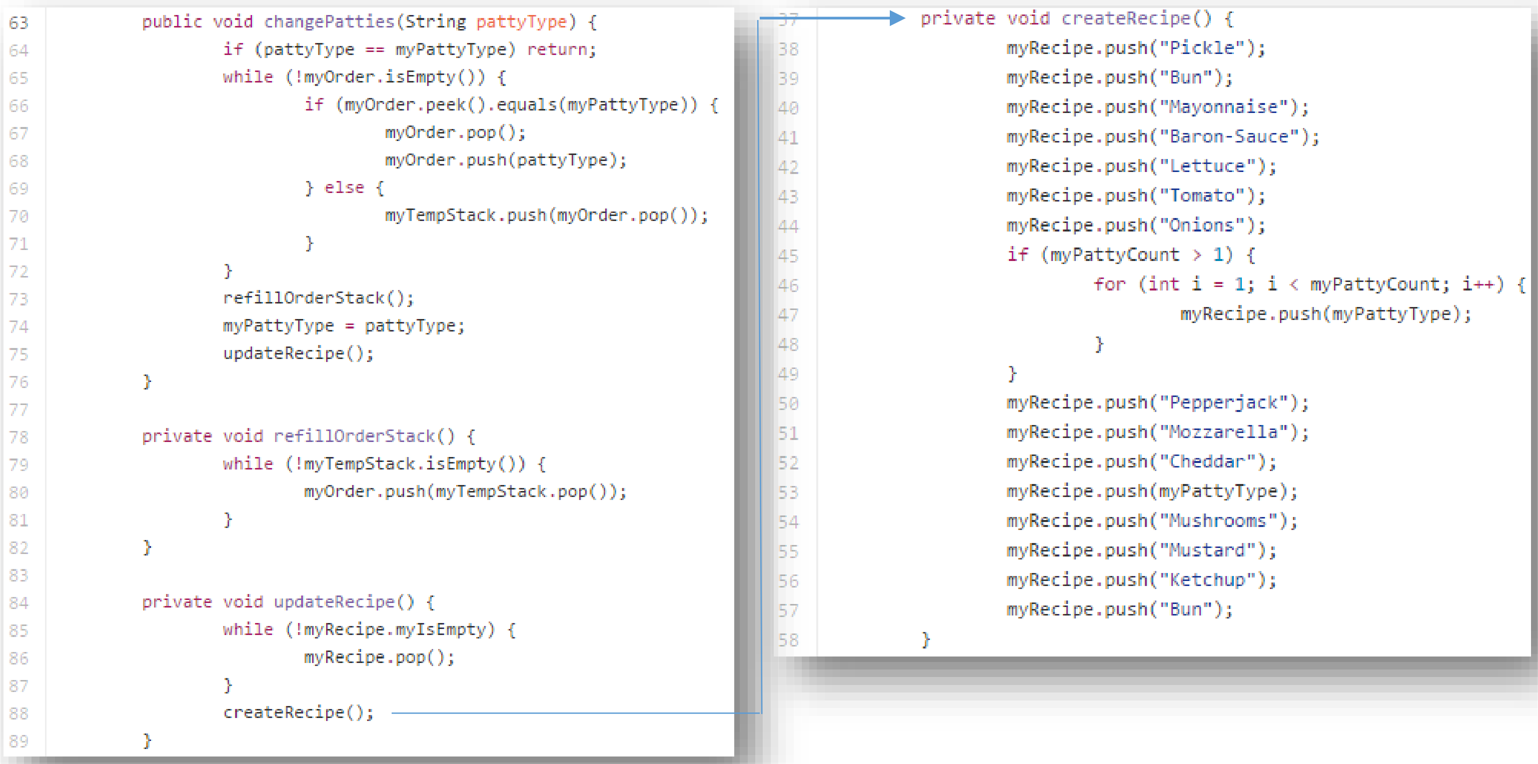
Line-By-Line



```
63      public void changePatties(String pattyType) {
64          if (pattyType == myPattyType) return;
65          while (!myOrder.isEmpty()) {
66              if (myOrder.peek().equals(myPattyType)) {
67                  myOrder.pop();
68                  myOrder.push(pattyType);
69              } else {
70                  myTempStack.push(myOrder.pop());
71              }
72          }
73          refillOrderStack();
74          myPattyType = pattyType;
75          updateRecipe();
76      }
77
78      private void refillOrderStack() {
79          while (!myTempStack.isEmpty()) {
80              myOrder.push(myTempStack.pop());
81          }
82      }
83
84      private void updateRecipe() {
85          while (!myRecipe.myIsEmpty) {
86              myRecipe.pop();
87          }
88          createRecipe();
89      }
```

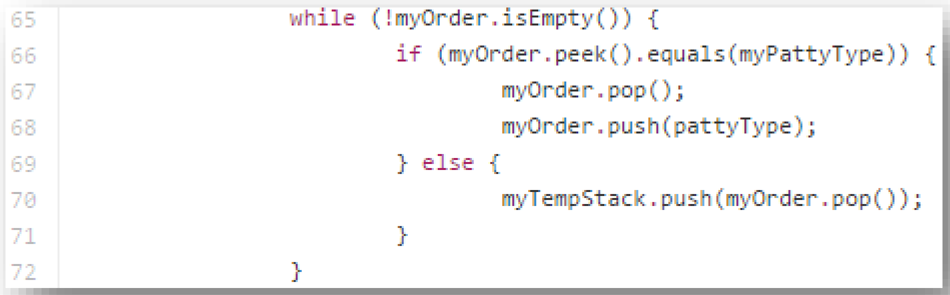
```
37      private void createRecipe() {
38          myRecipe.push("Pickle");
39          myRecipe.push("Bun");
40          myRecipe.push("Mayonnaise");
41          myRecipe.push("Baron-Sauce");
42          myRecipe.push("Lettuce");
43          myRecipe.push("Tomato");
44          myRecipe.push("Onions");
45          if (myPattyCount > 1) {
46              for (int i = 1; i < myPattyCount; i++) {
47                  myRecipe.push(myPattyType);
48              }
49          }
50          myRecipe.push("Pepperjack");
51          myRecipe.push("Mozzarella");
52          myRecipe.push("Cheddar");
53          myRecipe.push(myPattyType);
54          myRecipe.push("Mushrooms");
55          myRecipe.push("Mustard");
56          myRecipe.push("Ketchup");
57          myRecipe.push("Bun");
58      }
```

Line	Code	Big-Oh
Line 64	`if (pattyType == myPattyType)`	O(1)
Line 65	`!myOrder.isEmpty()`	O(1)
Line 66	`myOrder.peek().equals(myPattyType))`	O(3)
Line 67	`myOrder.pop()`	O(3)
Line 68	`myOrder.push(pattyType)`	O(7)
Line 70	`myTempStack.push(myOrder.pop())`	O(10)
Line 73	`refillOrderStack()`	O(n)
Line 74	`myPattyType = pattyType`	O(1)
Line 88	`createRecipe()`	O(n)

1. Line 64: This is a comparison operation which is 1.
2. Line 65: This is an access operation which is C_1.
3. Line 66: This line has a `peek` operation that is being used in a comparison. Together, these are $C_2 + 1$.
4. Line 67: This line has a call to the method `pop` which is C_3.
5. Line 68: This line has a call to the `push` method which is C_4.
6. Line 70: This line has a call to the `push` and `pop` methods which are $C_3 + C_4$.
7. Line 73: This line has a method call to `refillOrderStack()` which subsequently contains a `while` loop that checks to see if the stack is not empty and if true, a `push`/`pop` operation is performed. Together, the conditional check, and `push`/`pop` operations make $n(C_3 + C_4 + 1)$.
8. Line 74: This is a comparison operation which is 1.
9. Line 88: This line has a method call to `createRecipe()` which `push`(n ingredients) onto the stack. Therefore, we have $n \cdot C_{4.}$

Loops

The main `while` loop:



```
65  while (!myOrder.isEmpty()) {
66          if (myOrder.peek().equals(myPattyType)) {
67                  myOrder.pop();
68                  myOrder.push(pattyType);
69          } else {
70                  myTempStack.push(myOrder.pop());
71          }
72  }
```

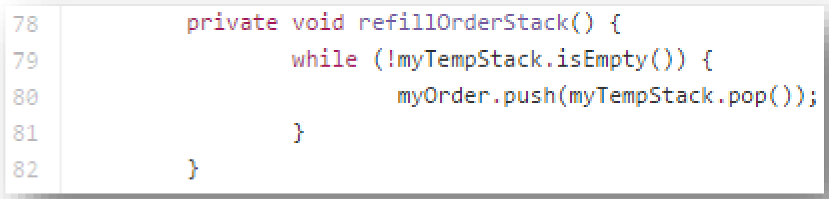
The first `while` loop consists of lines 65 through 70. The loop will terminate when the order stack is empty. The worst case occurs when there are two extra patties that need to be evaluated.

Let $f(n)$ be a function expressing the total cost of the `while` loop. We can express this sum as:

$$
\begin{aligned}
f(n) &= \sum_{i=0}^{n-1} (1 + C_2 + C_3 + C_4) \\
&= (1 + C_2 + C_3 + C_4) \sum_{i=0}^{n-1} 1 \\
&= (1 + C_2 + C_3 + C_4) \cdot (n - 1)
\end{aligned}
$$

Let $C_5 = (1 + C_2 + C_3 + C_4)$. Then the total cost of the `while` loop is $C_5 \cdot (n - 1)$.

The `refillOrderStack()` loop:



```
78        private void refillOrderStack() {
79                while (!myTempStack.isEmpty()) {
80                        myOrder.push(myTempStack.pop());
81                }
82        }
```

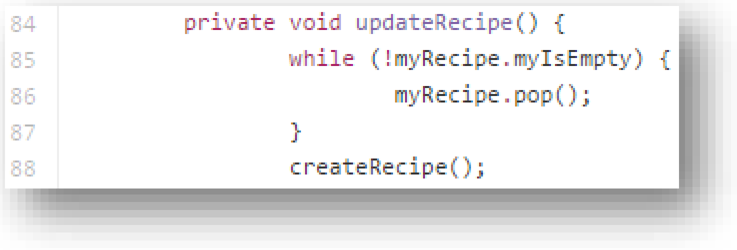
The next loop is contained within lines 79 to 80. The loop will terminate when the temp stack is empty. The worst case is n.

Let $f(n)$ be a function expressing the total cost of the `while` loop. We can express this sum as:

$$\begin{aligned} f(n) &= \sum_{i=0}^{n-1}(1 + C_3 + C_4) \\ &= (1 + C_3 + C_4)\sum_{i=0}^{n-1} 1 \\ &= (1 + C_3 + C_4)\cdot(n-1) \end{aligned}$$

Let $C_5 = (1 + C_3 + C_4)$. Then the total cost of the `while` loop is $C_5 \cdot (n-1)$.

The updateRecipe() loop:



```
84          private void updateRecipe() {
85                  while (!myRecipe.myIsEmpty) {
86                          myRecipe.pop();
87                  }
88                  createRecipe();
```

The next loop is contained within lines 85 to 86. The loop will terminate when the recipe stack is empty. The worst case is n.

Let $f(n)$ be a function expressing the total cost of the `while` loop. We can express this sum as:

$$\begin{aligned} f(n) &= \sum_{i=0}^{n-1}(1 + C_4) \\ &= (1 + C_4)\sum_{i=0}^{n-1} 1 \\ &= (1 + C_4) \cdot (n - 1) \end{aligned}$$

Let $C_5 = (1 + C_4)$. Then the total cost of the `while` loop is $C_5 \cdot (n - 1)$.

Total Cost

$$g(n) = 1 + 1 + 17 \cdot C_3 + \sum_{i=0}^{n-1}(1 + C_2 + C_3 + C_4) + \sum_{i=0}^{n-1}(1 + C_3 + C_4) + \sum_{i=0}^{n-1}(1 + C_4)$$

$$= (2 + 17 \cdot C_3) + (1 + C_2 + C_3 + C_4)\sum_{i=0}^{n-1} 1 + (1 + C_3 + C_4)\sum_{i=0}^{n-1} 1 + (1 + C_4)\sum_{i=0}^{n-1} 1$$

$$= (2 + 17 \cdot C_3 + (1 + C_2 + C_3 + C_4) + (1 + C_3 + C_4) + (1 + C_4))\sum_{i=0}^{n-1} 1 \sum_{i=0}^{n-1} 1 \sum_{i=0}^{n-1} 1$$

$$= (2 + 17 \cdot C_3 + (1 + C_2 + C_3 + C_4) + (1 + C_3 + C_4) + (1 + C_4)) \cdot 3(n - 1)$$

$$= (C_2 + 19 \cdot C_3 + 4 \cdot C_4 + 5) \cdot 3(n - 1)$$

Let $a = (C_2 + 19 \cdot C_3 + 4 \cdot C_4 + 5)$ and $b = 3$, therefore, $g(n) = a + b(n)$.

We can see that $g(n) \in O(n)$.