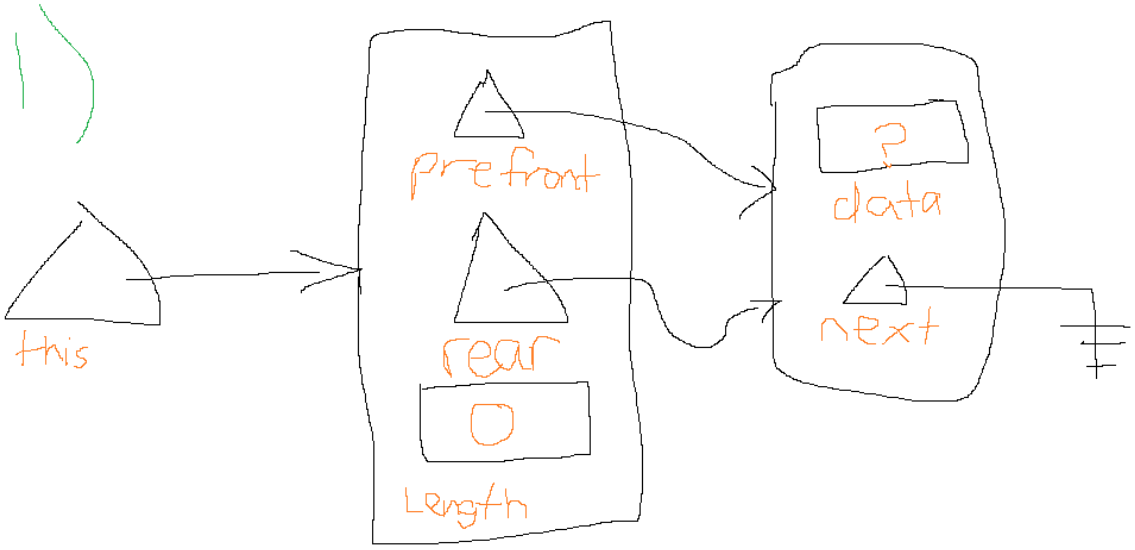
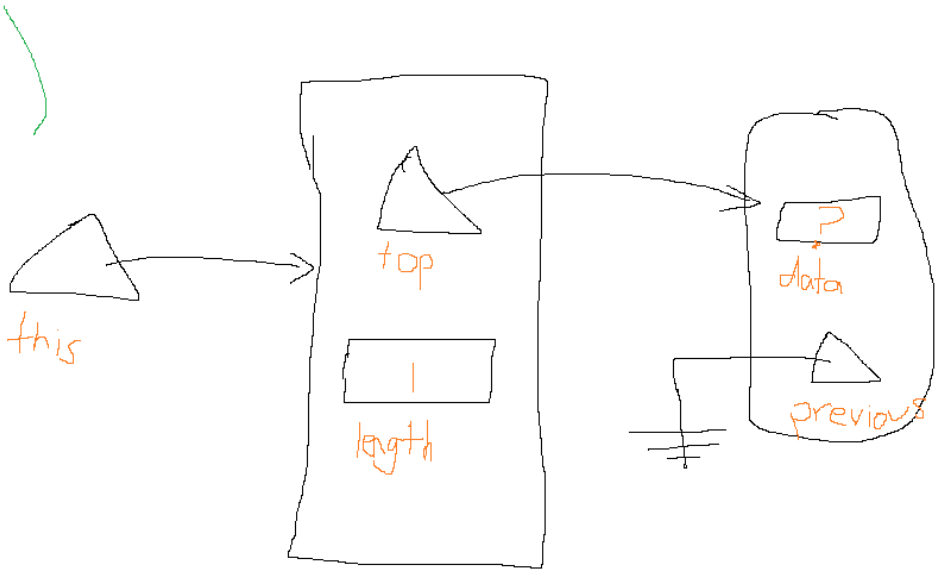
1)


this
prefront
rear
0
Length
?
data
next

2)


this
top
1
length
?
data
previous

```
@Override
public final void push(T x) {
assert x != null : "Violation of: x is not null";
// create new node
Node temp = new Node();
temp.data = x;
// rearrange nodes
temp.next = this.top;
this.top = temp;
this.length++;
assert this.conventionHolds();
}
@Override
public final T pop() {
assert this.length() > 0 : "Violation of: this /= <>";
T temp = this.top.data;
this.top = this.top.next;
this.length--;
assert this.conventionHolds();
// Fix this line to return the result after checking the convention.
return temp;
}
@Override
public final int length() {
assert this.conventionHolds();
```

```
// Fix this line to return the result after checking the convention.
return this.length;
}
```

```
@Test
public void constructorTest1() {
Stack<Object> test = new Stack2<>();
Stack<Object> ref = new Stack2<>();
assertEquals(ref, test);
}
@Test
public void pushTest1() {
Stack<Object> test = new Stack2<>();
Stack<Object> ref = new Stack2<>();
test.push(4);
test.push(5);
test.push(6);
ref.push(4);
ref.push(5);
ref.push(6);
assertEquals(ref, test);
}
@Test
```

```
public void popTest1() {
Stack<Object> test = new Stack2<>();
Stack<Object> ref = new Stack2<>();
test.push(4);
test.push(5);
test.push(6);
ref.push(4);
ref.push(5);
ref.push(6);
test.pop();
ref.pop();
assertEquals(ref, test);
test.pop();
ref.pop();
assertEquals(ref, test);
test.pop();
ref.pop();
test.pop();
ref.pop();
assertEquals(ref, test);
}
@Test
public void lengthTest() {
Stack<Object> test = new Stack2<>();
Stack<Object> ref = new Stack2<>();
```

```
test.push(4);

test.push(5);

test.push(6);

ref.push(4);

ref.push(5);

ref.push(6);

assertEquals(ref.length(), test.length());

}

}
```