Deque

Ja-Hee Kim

Agenda

01 ADT

Deque

02 Examples

Computing the Capital Gain in a Sale of Stock,

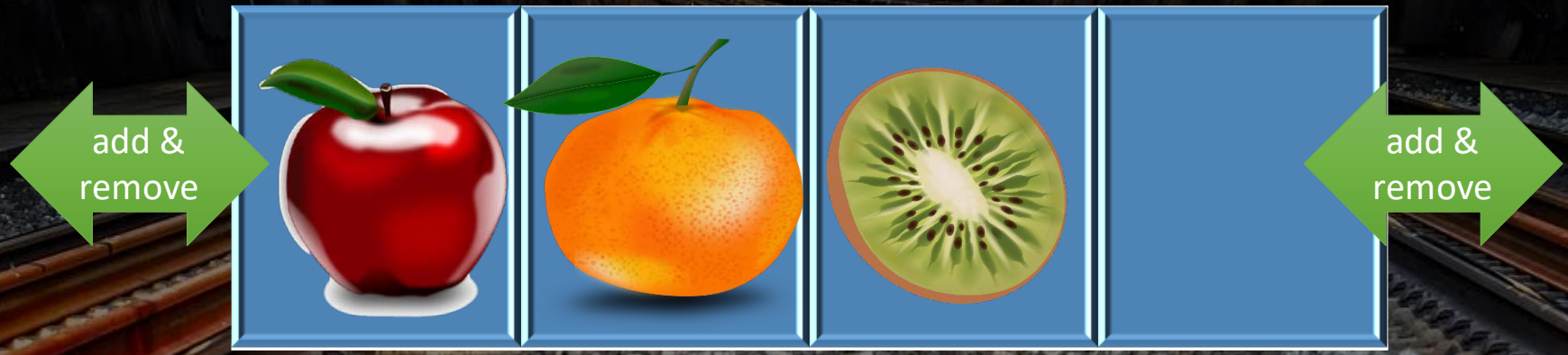
03 Linked Deque

Enqueue & dequeue

ADT of Deque

Deque

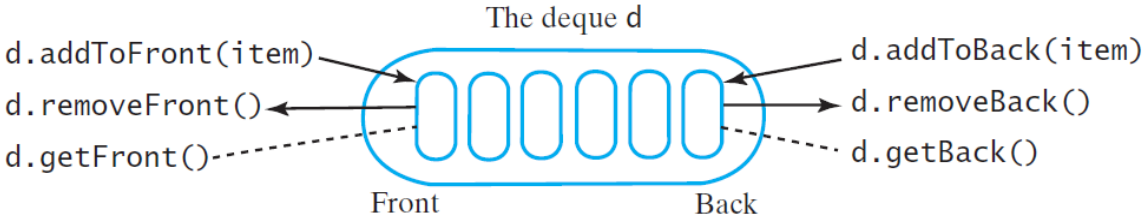
- Double-ended queue
- Pronounced “deck”
- generalizes a queue, for which elements can be added to or removed from either the fron t (head) or back (tail).



Deque

operation	common name	C++	Java	Perl	PHP	Python	JavaScript
insert element at back	inject, snoc, push	push_back	offerLast addLast add,offer(Q)	push	array_push	append	push
insert element at front	push, cons	push_front	offerFirst addLast push(stack)	unshift	array_unshift	appendleft	unshift
remove last element	eject	pop_back	pollLast removeFirst pop(stack)	pop	array_pop	pop	pop
remove first element	pop	pop_front	pollFirst removeFirst remove,poll(Q)	shift	array_shift	popleft	shift
examine last element	peek	back	peekLast getLast	$array[-1]	end	<obj>[-1]	<obj>[<obj>.length - 1]
examine first element		front	peekFirst getFirst peek,element	$array[0]	reset	<obj>[0]	<obj>[0]

Interface of a Deque



Deque<T>
+addToFront(newEntry): void +removeFront():T +getFront():T +addToBack(newEntry): void +removeBack():T +getBack():T

```java
/**
 * An Interface for the ADT deque
 * @author jahee
 * @param <T> : the class type that a dequeue constrains
 */
public interface DequeInterface<T> {
    /** Add a new etnry to the front of this deque
     * @param newEntry an Object to be added
     */
    public void addToFront(T newEntry);
    /** Add a new etnry to the back of this deque
     * @param newEntry an Object to be added
     */
    public void addToBack(T newEntry);
    /**
     * Removes and returns the front entry of this deque.
     * @return the object at the front of the deque
     */
    public T removeFront();
    /**
     * Removes and returns the back entry of this deque.
     * @return the object at the back of the deque
     */
    public T removeBack();
    /**
     * Retrieves the front entry of this deque.
     * @return the object at the front of the deque
     */
    public T getFront();
    /**
     * Retrieves the back entry of this deque.
     * @return the object at the back of the deque
     */
    public T getBack();
    /**
     * Detects whether this deque is empty
     * @return true if the deque is empty. or false otherwise
     */
    public boolean isEmpty();
    /**
     * removes all entries from this deque
     */
    public void clear();
}
```

Build-in library, interface Deque

Build-in library, Class ArrayDeque

java.util

Class ArrayDeque<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.ArrayDeque<E>

Type Parameters:

`E` - the type of elements held in this collection

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, Deque<E>, Queue<E>

```
public class ArrayDeque<E>
extends AbstractCollection<E>
implements Deque<E>, Cloneable, Serializable
```

Resizable-array implementation of the `Deque` interface. Array deques have no capacity restrictions; they grow as necessary to support usage. They are not thread-safe; in the absence of external synchronization, they do not support concurrent access by multiple threads. Null elements are prohibited. This class is likely to be faster than `Stack` when used as a stack, and faster than `LinkedList` when used as a queue.

Most `ArrayDeque` operations run in amortized constant time. Exceptions include `remove`, `removeFirstOccurrence`, `removeLastOccurrence`, `contains`, `iterator.remove()`, and the bulk operations, all of which run in linear time.

The iterators returned by this class's `iterator` method are *fail-fast*: If the deque is modified at any time after the iterator is created, in any way except through the iterator's own `remove` method, the iterator will generally throw a `ConcurrentModificationException`. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw `ConcurrentModificationException` on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: *the fail-fast behavior of iterators should be used only to detect bugs.*

This class and its iterator implement all of the *optional* methods of the `Collection` and `Iterator` interfaces.

This class is a member of the Java Collections Framework.

Since:

1.6

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor and Description
`ArrayDeque()` Constructs an empty array deque with an initial capacity sufficient to hold 16 elements.
`ArrayDeque(Collection<? extends E> c)` Constructs a deque containing the elements of the specified collection, in the order they are returned by the collection's iterator.
`ArrayDeque(int numElements)` Constructs an empty array deque with an initial capacity sufficient to hold the specified number of elements.

Examples of Deque

Computing the Capital Gain in a Sale of Stock

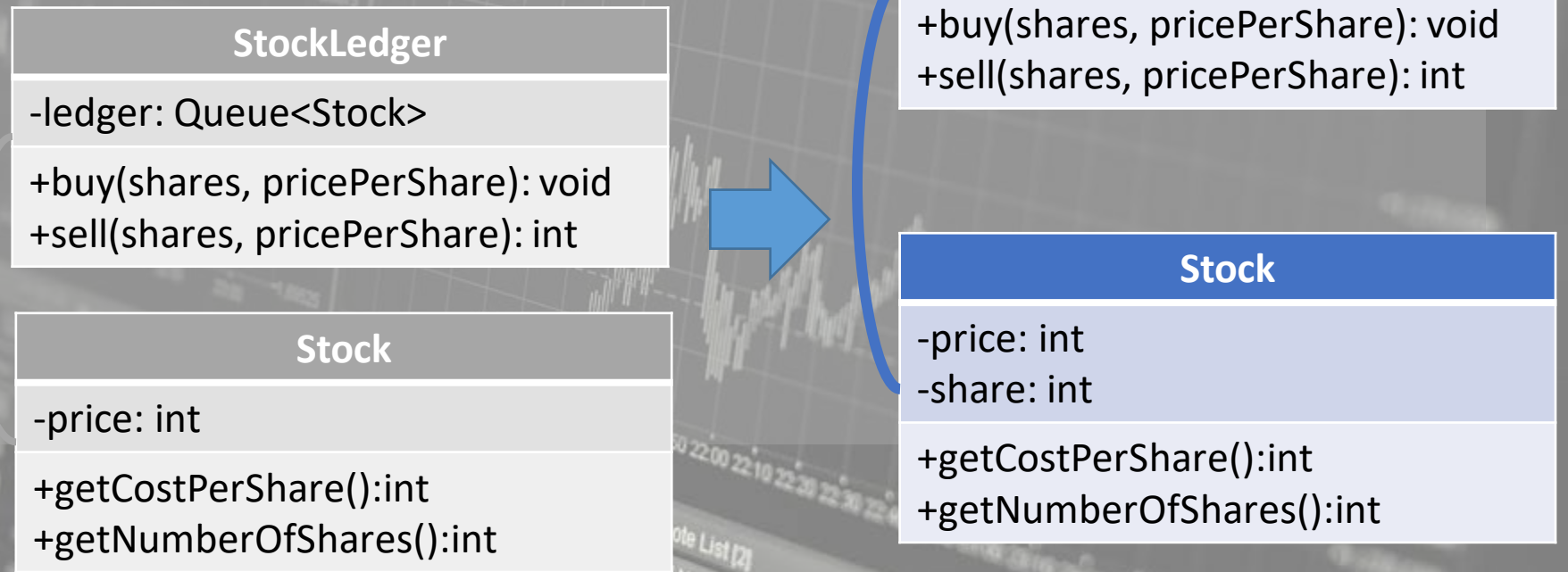
- Capital gain: a profit that you have made if the sale price exceeds the purchase price
  - stock sales are a first-in, first-out application
- Example: Presto Pizza
  - Last year: buy 6/$45
  - Last month; buy 6/$75
  - Today: sell 9/$65

6/$65-$45	=6*$20	=$120
3/$65-$75	=3*-$10	=-$30
		$90

Design of Problem

- Responsibility of StockLedger:
  - Record the shares of a stock purchased, in chronological order
  - Remove the shares of a stock sold, beginning with the ones held the longest
  - Compute the capital gain (loss) on shares of a stock sold

- Responsibility of Stock
  - Getter method for two private number



Computing the Capital Gain in a Sale of Stock

```java
public class StockMain {
    public static void main(String[] args) {
        StockLedger book = new StockLedger();
        book.buy(6,45);
        book.buy(6,75);
        System.out.println("My capital gain is: "+book.sell(9, 65));
    }
}
```

Problems Javadoc Declaration Console

<terminated> StockMain [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe (2020. 10. 6. 오전 2:19:49 – 오전 2:19:51)

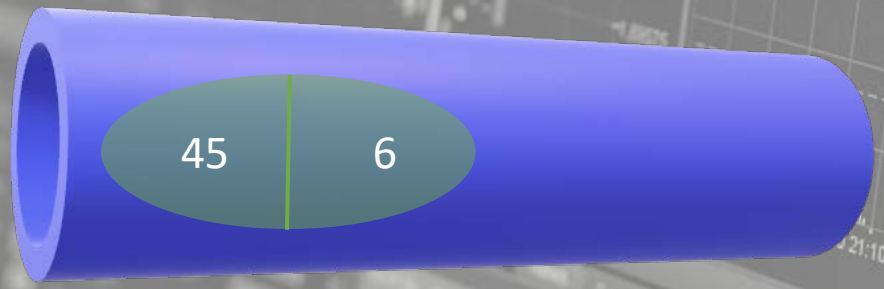
```
My capital gain is: 90.0
```

buy

Client program

book.buy(6,45);

```
import java.util.*;
/** Records the purchase and sale of stocks,
 * and provides the capital gain or loss. */
public class StockLedger{
    private Deque<Stock> ledger;
    public StockLedger()    {
        ledger = new ArrayDeque<Stock>(100);
    }
    /** Records a stock purchase in this ledger.
     * @param sharesBought the number of shares purchased
     * @param pricePerShare the price per share */
    public void buy(int sharesBought, int pricePerShare)    {
        ledger.offer(new Stock(pricePerShare, sharesBought));
    }
```

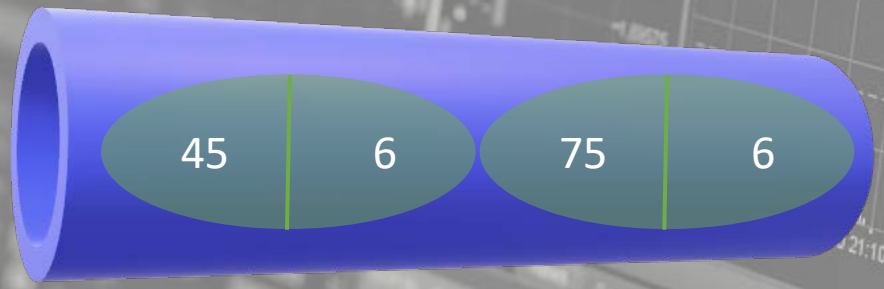


buy

Client program

book.buy(6,75);

```
import java.util.*;
/** Records the purchase and sale of stocks,
 * and provides the capital gain or loss. */
public class StockLedger{
    private Deque<Stock> ledger;
    public StockLedger()    {
        ledger = new ArrayDeque<Stock>(100);
    }
    /** Records a stock purchase in this ledger.
     * @param sharesBought the number of shares purchased
     * @param pricePerShare the price per share */
    public void buy(int sharesBought, int pricePerShare)    {
        ledger.offer(new Stock(pricePerShare, sharesBought));
    }
```

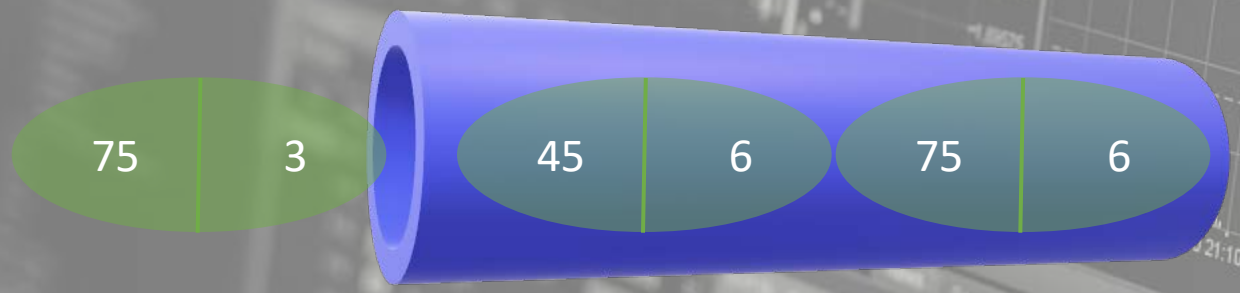


sell

Client program

System.*out*.println("My capital gain is: "+book.sell(9, 65));

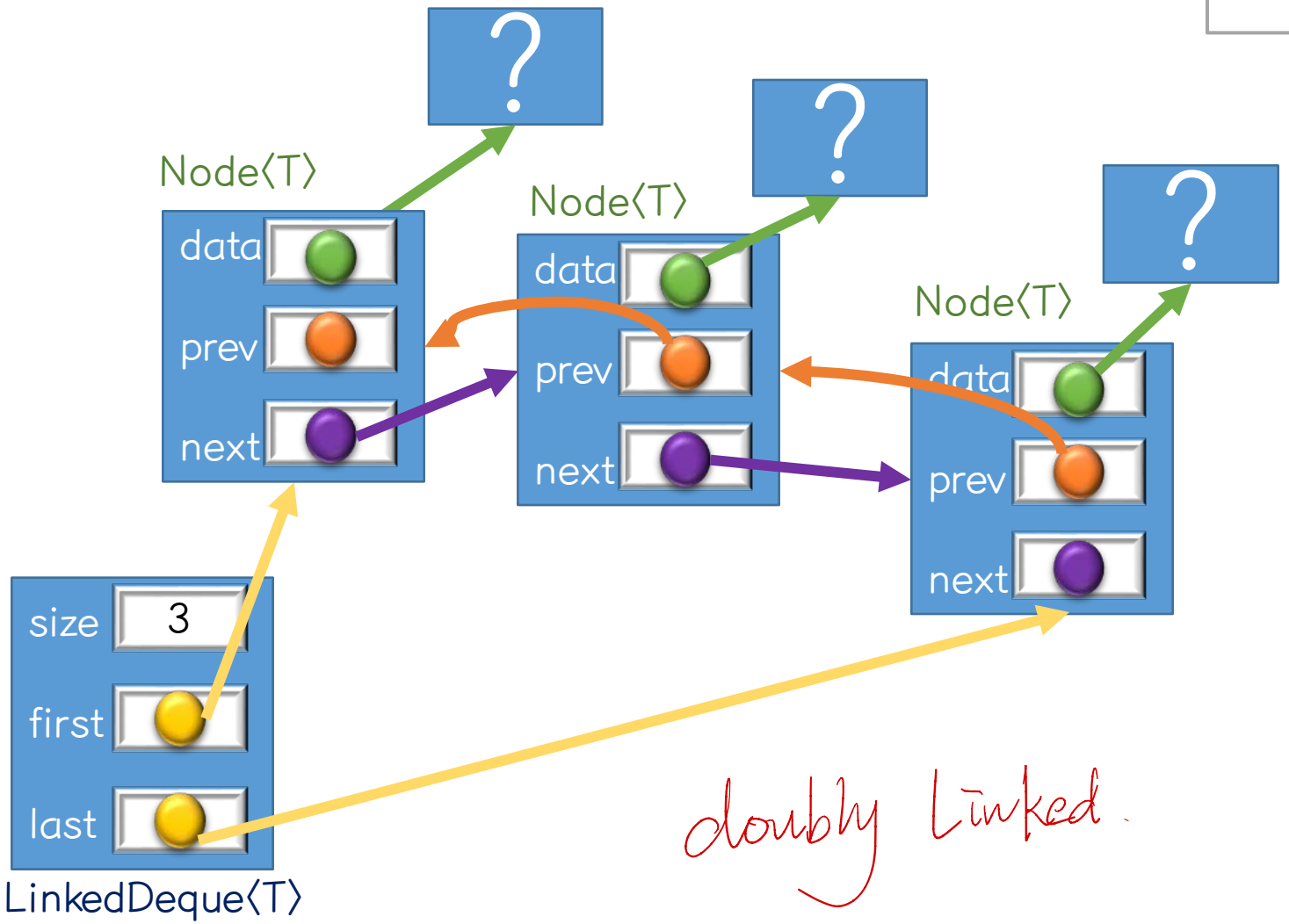
```
public double sell(int sharesSold, int pricePerShare)   {
    int saleAmount = sharesSold * pricePerShare;
    int totalCost = 0;
    while(sharesSold > 0)       {
        Stock transaction = ledger.removeFirst();
        int stockCost = transaction.getCostPerShare();
        int stockNumber = transaction.getNumberOfShares();
        if(stockNumber > sharesSold) {
            totalCost = totalCost + sharesSold*stockCost;
            ledger.addFirst(new Stock(stockCost, stockNumber - sharesSold));
        } else
            totalCost = totalCost + stockCost*stockNumber;
        sharesSold = sharesSold - stockNumber;
    } // end while
    return saleAmount - totalCost; // gain or loss
} // end sell
```



sharesSold: 9
totalCost: 0
sharesSold: 9
salesAmount: 9 x 65

Doubly Linked Implementation of a Deque

Outline of the class



the size of the Deque

LinkedDeque<T>

- size: int
- first: Node<T>
- last: Node<T>

+addToFront(newEntry): void
+removeFront():T
+getFront():T
+addToBack(newEntry): void
+removeBack():T
+getBack():T
+size():int
+toString(): String

Node<T>

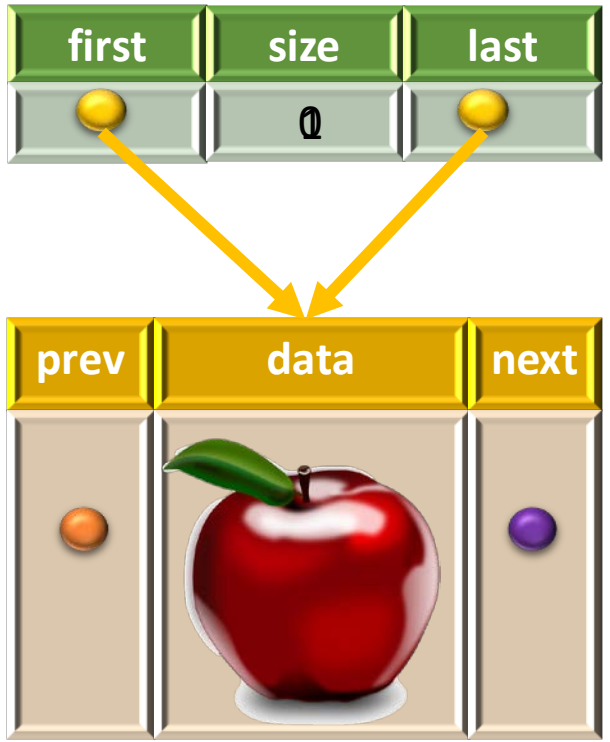
- data: T
- next: Node<T>
- prev:Node<T>

+ Node(T data, , Node p, Node n)

Method addToFront

- Client program

crate.addToFront("Apple");



- In class LinkedDeque

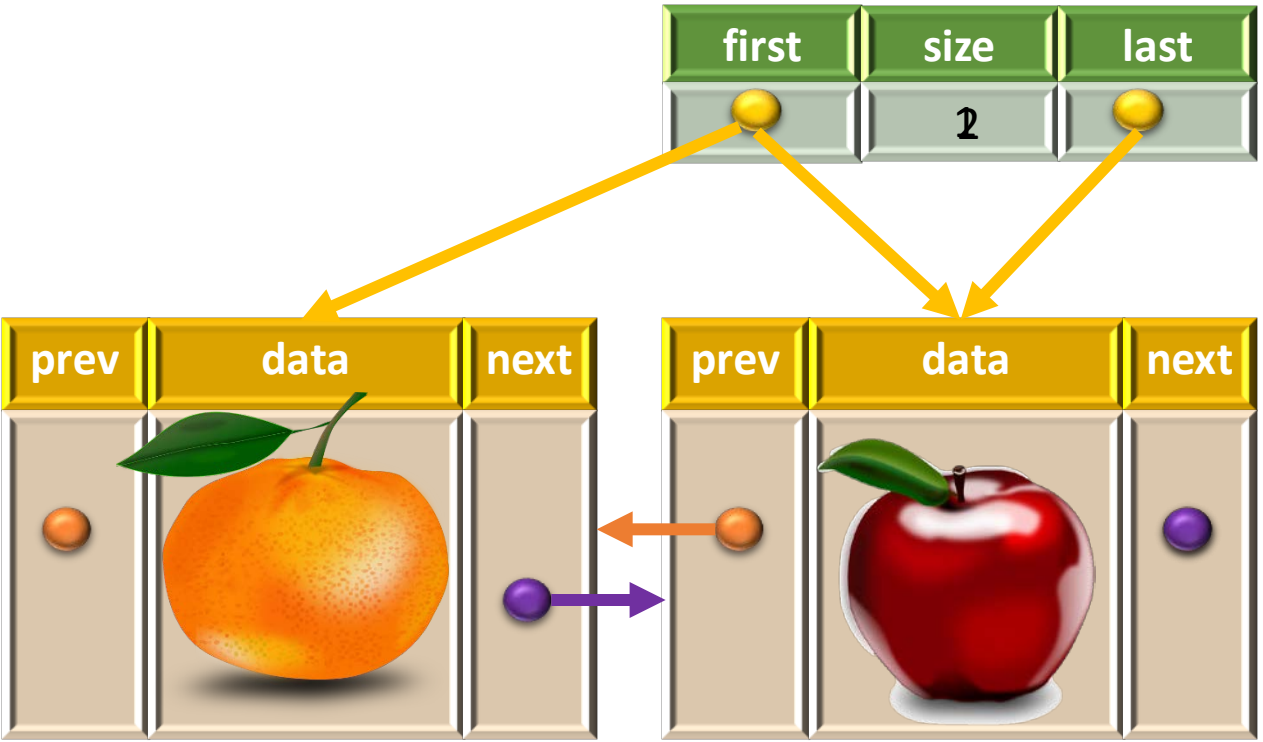
```
public void addToFront(T newEntry) {
   if(isEmpty())
      first = last = new Node(newEntry, null, first);
   else
      first = first.prev
            = new Node(newEntry, null, first);
   size++;
}
```

O(1)

Method addToFront

- Client program

crate.addToFront("Orange");



- In class LinkedDeque

```
public void addToFront(T newEntry) {
   if(isEmpty())
      first = last = new Node(newEntry, null, first);
   else
      first = first.prev
            = new Node(newEntry, null, first);
   size++;
}

O(1)
```

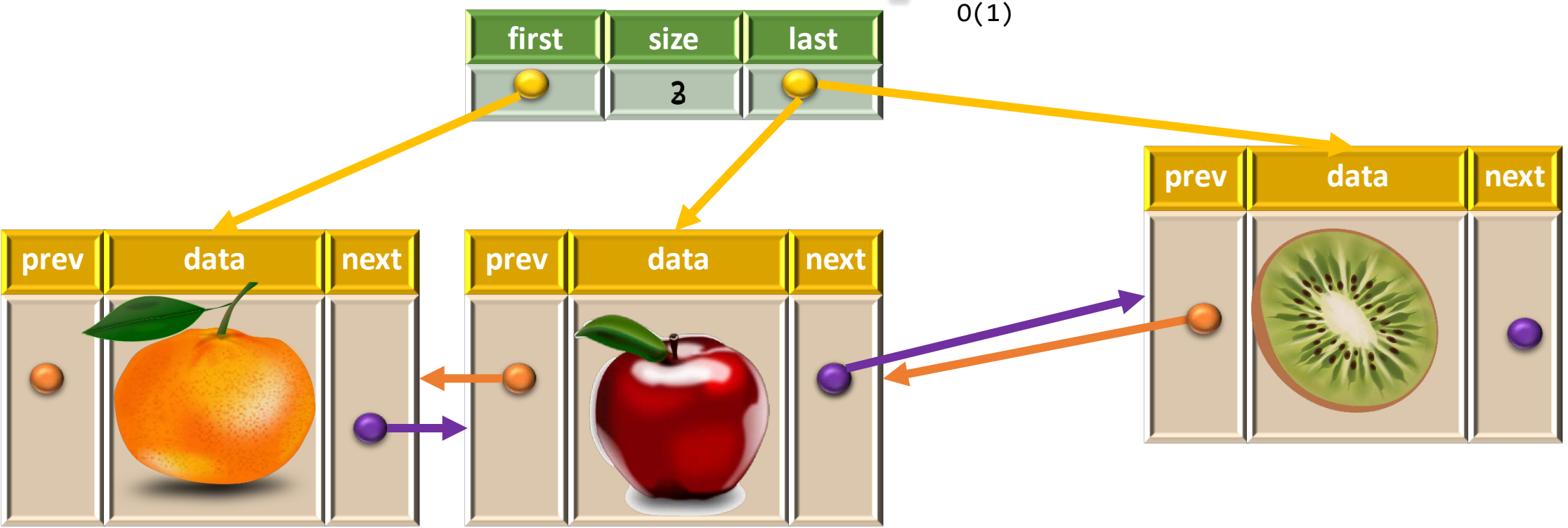
Method addToBack

- Client program

crate.addToBack("Kiwi");

- In class LinkedDeque

```
public void addToBack(T newEntry) {
   if(isEmpty())
      last = first = new Node(newEntry, last, null);
   else
      last = last.next = new Node(newEntry, last, null);
   size ++;
}
O(1)
```



Method getFront & getBack

```
1  public class DequeTest {
2      public static void main(String[] args) {
3          DequeInterface<String> crate = new LinkedDeque();
4          crate.addToFront("Apple");
5          crate.addToFront("Orange");
6          crate.addToBack("Kiwi");
7          System.out.println("getFront: "+crate.getFront());
8          System.out.println("getBack: "+crate.getBack());
9      }
10 }
```

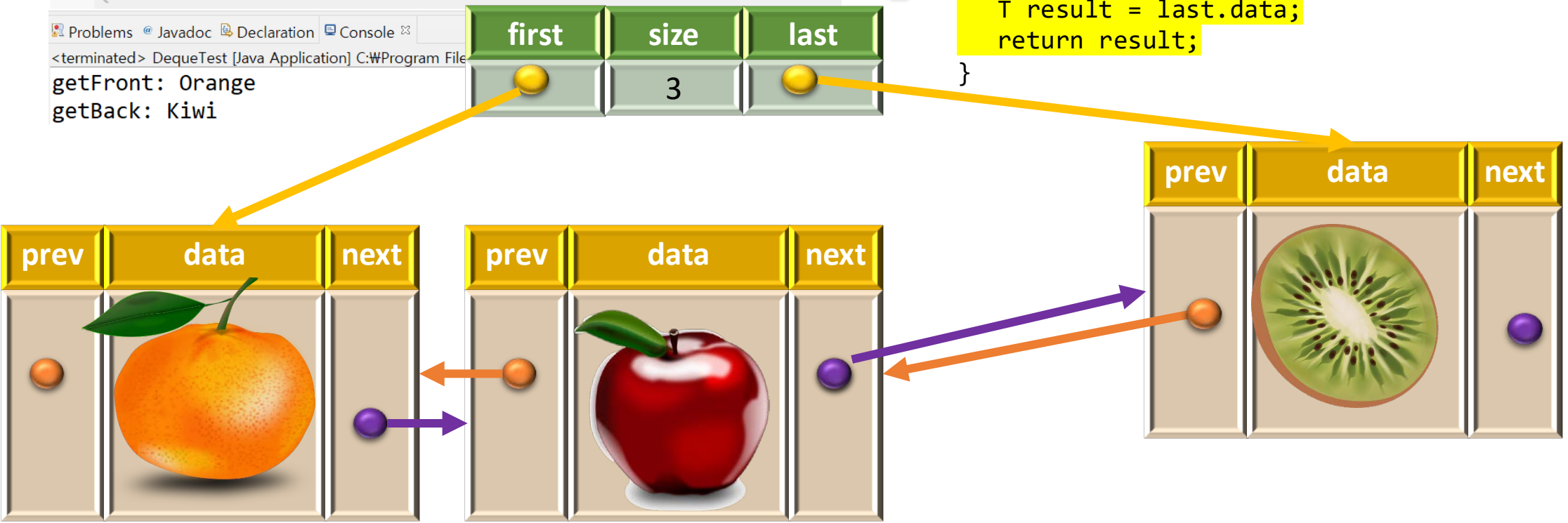
Problems Javadoc Declaration Console

<terminated> DequeTest [Java Application] C:₩Program File

```
getFront: Orange
getBack: Kiwi
```

- In class LinkedDeque

```
public T getFront() {
   if(first == null) return null;
   T result = first.data;
   return result;
}
public T getBack() {
   if(last == null) return null;
   T result = last.data;
   return result;
}
```



Method removeFront

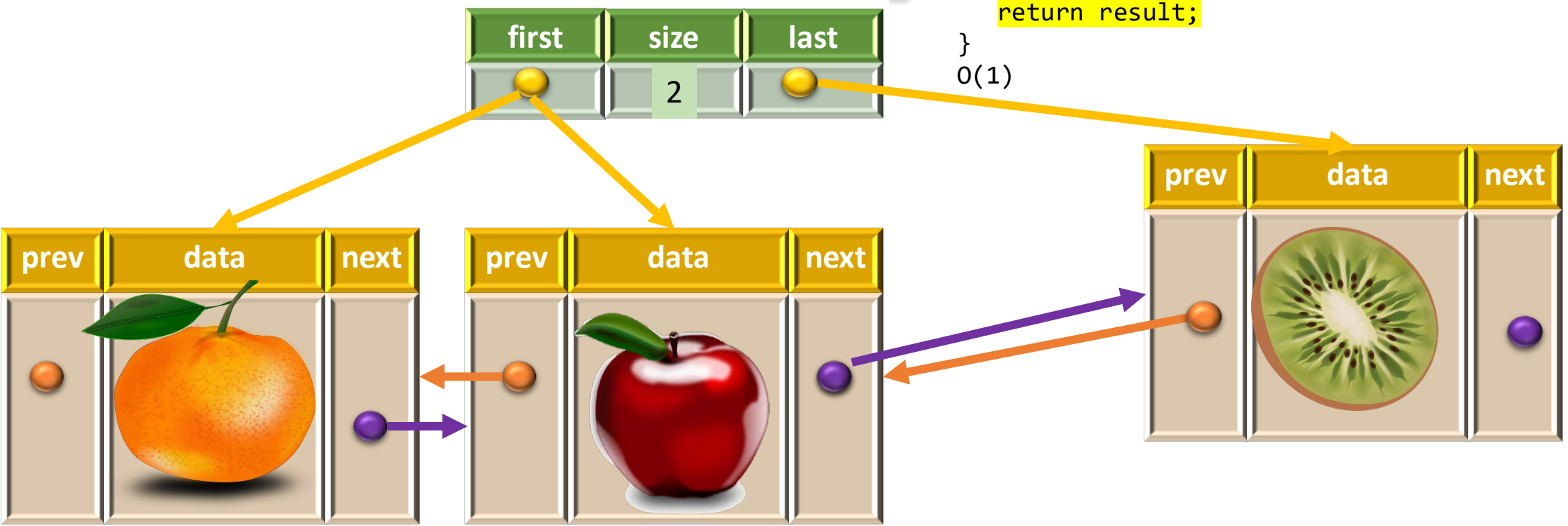
```
 9        System.out.println("after removing Front: "+crate.removeFront());
10        System.out.println(crate.toString()+", "+((LinkedDeque)crate).reverseString());
11    }
12 }
```

Problems Javadoc Declaration Console

```
<terminated> DequeTest [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe (2020. 10. 8. 오후 10:08:09 – 오후 10:08:11)
getFront: Orange
getBack: Kiwi
after removing Front: Orange
Deque[2]:Apple Kiwi , Reverse Deque[2]:Kiwi Apple
```

- In class LinkedDeque

```
public T removeFront() {
    T result = getFront();
    if (result == null) return null; # of element ==0
    size --;
    first = first.next;
    if(first == null) last = null; # of element ==1
    else first.prev = null;
    return result;
}
O(1)
```



Method removeBack

```
 8          System.out.println("getBack: "+crate.getBack());
 9          System.out.println("after removing Front: "+crate.removeFront());
10          System.out.println(crate.toString()+", "
11                  +((LinkedDeque)crate).reverseString());
12          System.out.println("after removing Back: "+crate.removeBack());
13          System.out.println(crate.toString()+", "
14                  S+((LinkedDeque)crate).reverseString());
15      }
16  }
```

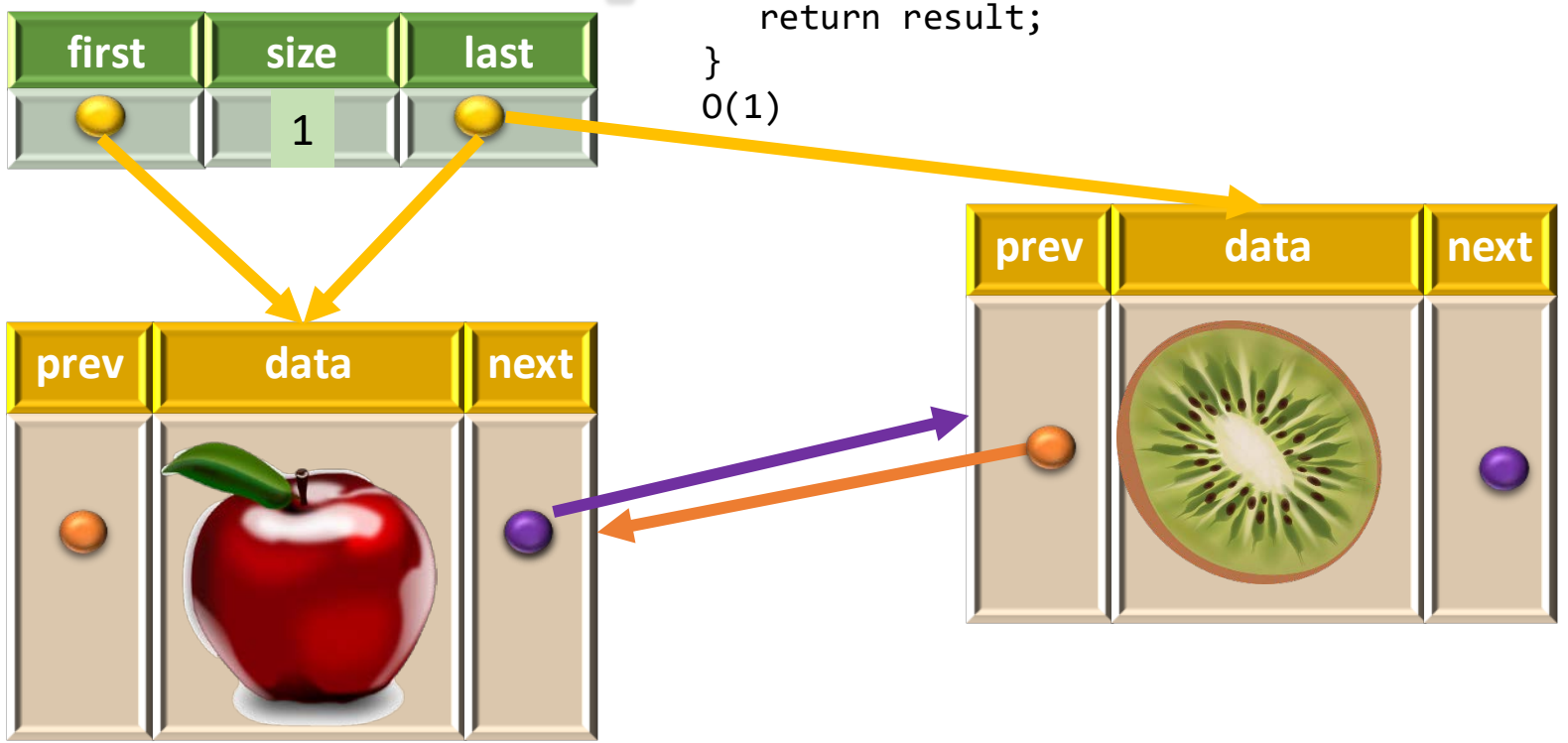
Problems Javadoc Declaration Console

<terminated> DequeTest [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe (2020. 10. 8. 오후 10:17:18 – 오후 10:17:18)

```
getFront: Orange
getBack: Kiwi
after removing Front: Orange
Deque[2]:Apple Kiwi , Reverse Deque[2]:Kiwi Apple
after removing Back: Kiwi
Deque[1]:Apple , Reverse Deque[1]:Apple
```

- In class LinkedDeque

```
public T removeBack() {
    T result = getBack();
    if (result == null) return null; # of element == 0
    size --;
    last = last.prev;
    if(last == null) first = null; # of element == 1
    else last.next = null;
    return result;
}
O(1)
```



Thank you