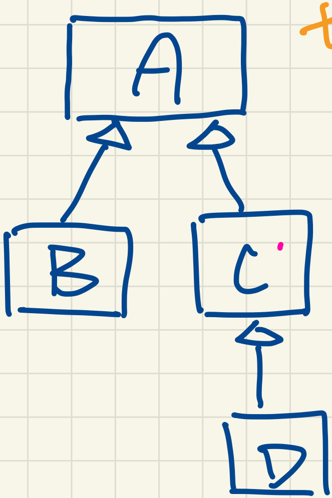
Lecture 14 - Wednesday, March 1

Announcements

- Makeup Lecture for WrittenTest1
  + Expected to complete by: March 20
- A2 solution: only source code (no solution videos)

Static Type vs. Dynamic Type

declared type



any object of type that's a class of A? a descendant

any descendent classes of the ST, including A itself

```
A oa = new C();
```

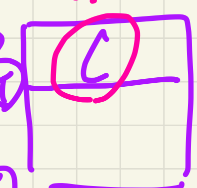
is valid

polymorphism

```
C oc = new D();
```

static types

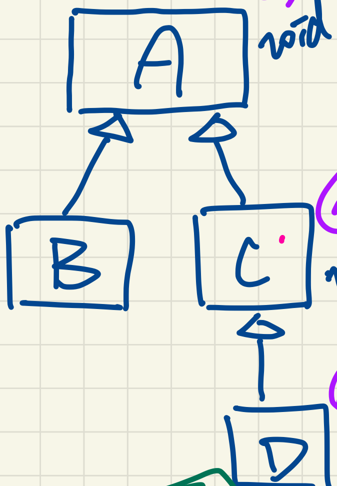
dynamic type



oa instanceof A (T)

oa instanceof C (T)

oa instanceof D (F)



(v1) void m(){...}

(v2) void m(){...} overridden impl.

(v3) void m(){...} overridden version

① A oa = new C();

② C oc = new D();

③ oa.m(); ~> invoke version of m in C (dynamic type)

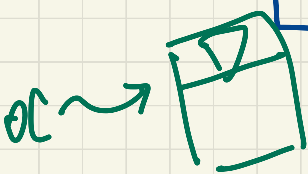
④ oa = oc;

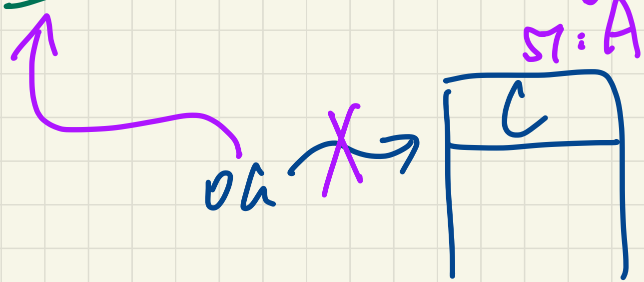
ST: A ST: C

oa.m(); → invoke version in D

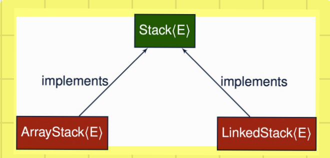
polymorphism

dynamic binding





Stack ADT: Testing Alternative Implementations



invoke the push imp. in AS class

```java
@Test
public void testPolymorphicStacks() {
  Stack<String> s = new ArrayStack<>();
  s.push("Alan"); /* dynamic binding */
  s.push("Mark"); /* dynamic binding */
  s.push("Tom"); /* dynamic binding */
  assertTrue(s.size() == 3 && !s.isEmpty());
  assertEquals("Tom", s.top());

  s = new LinkedStack<>();
  s.push("Alan"); /* dynamic binding */
  s.push("Mark"); /* dynamic binding */
  s.push("Tom"); /* dynamic binding */
  assertTrue(s.size() == 3 && !s.isEmpty());
  assertEquals("Tom", s.top());
}
```

ST

DT1

DT2

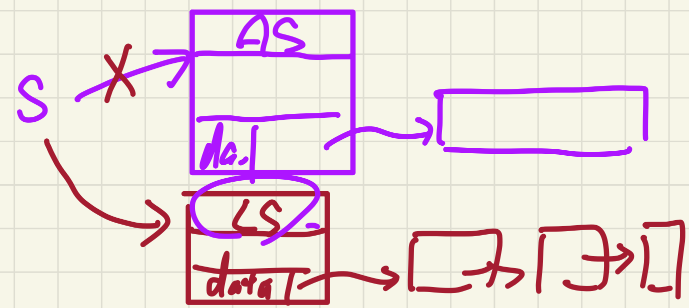
poly-morphism

```java
public class ArrayStack<E> implements Stack<E> {
  private final int MAX_CAPACITY = 1000;
  private E[] data;
  private t; /* index of top */
  public ArrayStack() {
    data = (E[]) new Object[MAX_CAPACITY];
    t = -1;
  }

  public int size() { return (t + 1); }
  public boolean isEmpty() { return (t == -1); }

  public E top() {
    if (isEmpty()) { /* Precondition Violated */ }
    else { return data[t]; }
  }
  public void push(E e) {
    if (size() == MAX_CAPACITY) { /* Precondition Violated */ }
    else { t ++; data[t] = e; }
  }
  public E pop() {
    E result;
    if (isEmpty()) { /* Precondition Violated */ }
    else { result = data[t]; data[t] = null; t --; }
    return result;
  }
}
```

invoke the push imp. in LS class



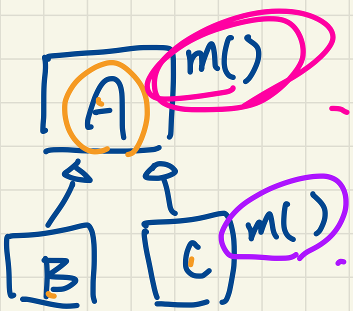
Polymorphic Collection (stack)

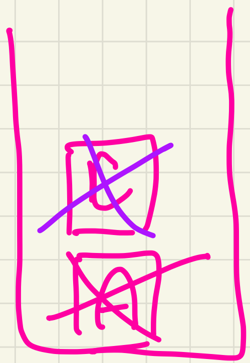
```
public interface Stack<E> {
  public int size();
  public boolean isEmpty();
  public E top();
  public void push(E e);
  public E pop();
}
```

Stack<A> s = new ...

* s.push(new A() / new B() / new C()); any descent classes of A

??





s.push(new A());

s.push(new C());

① A obj = s.pop(); ③ obj = s.pop();

DT: C   DT: A

② obj.m(); ④ obj.m();

Lecture

Stack ADT vs. Queue ADT

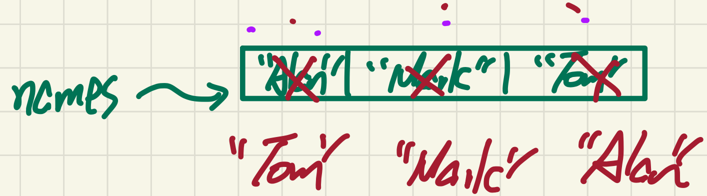
Stack ADT -
Algorithms using the Stack ADT

Algorithm using Stack: Reversing an Array

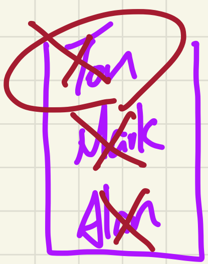
generic parameter declared at the method level.

```
public static <E> void reverse(E[] a) {
  Stack<E> buffer = new ArrayStack<E>();
  for (int i = 0; i < a.length; i ++) {
    buffer.push(a[i]);
  }
  for (int i = 0; i < a.length; i ++) {
    a[i] = buffer.pop();
  }
}
```

from L to R

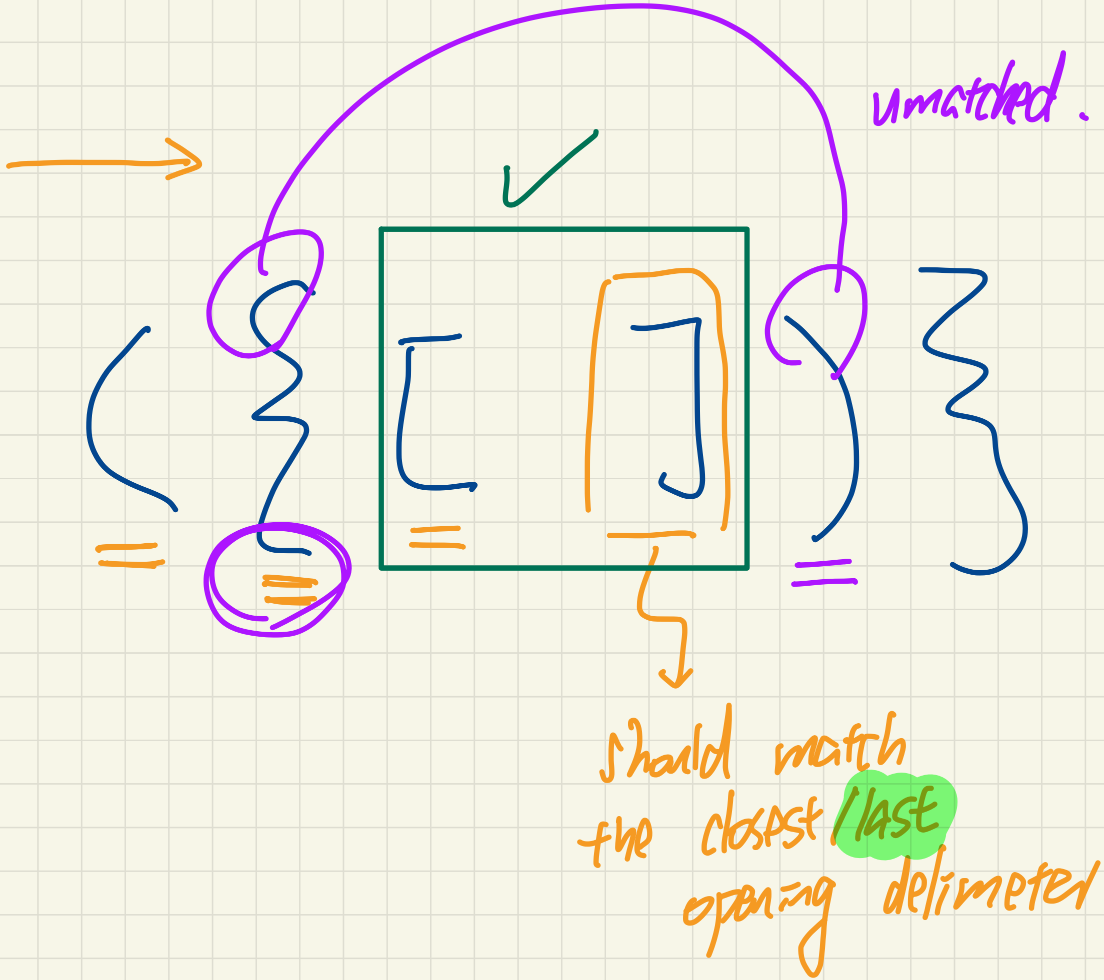


in-place reverse




```
@Test
public void testReverseViaStack() {
  String[] names = {"Alan", "Mark", "Tom"};
  String[] expectedReverseOfNames = {"Tom", "Mark", "Alan"};
  StackUtilities.reverse(names);
  assertArrayEquals(expectedReverseOfNames, names);

  Integer[] numbers = {46, 23, 68};
  Integer[] expectedReverseOfNumbers= {68, 23, 46};
  StackUtilities.reverse(numbers);
  assertArrayEquals(expectedReverseOfNumbers, numbers);
}
```


unmatched.
should match the closest/last opening delimeter

Algorithm using Stack: Matching Delimiters

```java
public static boolean isMatched(String expression) {
   final String opening = "([{";
   final String closing = ")]}";
   Stack<Character> openings = new LinkedStack<Character>();
   int i = 0;
   boolean foundError = false;
   while (!foundError && i < expression.length()) {
      char c = expression.charAt(i);
      if(opening.indexOf(c) != -1) { openings.push(c); }
      else if (closing.indexOf(c) != -1) {
         if(openings.isEmpty()) { foundError = true; }
         else {
            if (opening.indexOf(openings.top()) == closing.indexOf(c)) {
               openings.pop();
            }
            else { foundError = true; }
         }
      }
      i ++;
   }
   return !foundError && openings.isEmpty();
}
```

0 1 2

RT: O(n)

length of input string.

closing not matched by empty stack.

when closing matches open

closing not matched

openings may be non-empty

```java
@Test
public void testMatchingDelimiters() {
   assertTrue(StackUtilities.isMatched(""));
   assertTrue(StackUtilities.isMatched("{[]}({})"));
   assertFalse(StackUtilities.isMatched("{[])"));
   assertFalse(StackUtilities.isMatched("{[}"));
   assertFalse(StackUtilities.isMatched("({[]}"));
}
```

openings

openings

error

openings

has no matching top?

openings

end of input but stack is not empty

Post-fix notation

operands first, then operator.

$\boxed{3}\ \boxed{4\ 5\ *}\ -$

$3\ \boxed{4\ 5\ -}\ *$

$3 * (4 - 5)$

$\boxed{3\ 4\ -}\ 5\ *$

$(3 - 4) * 5$

$5\ \boxed{3\ 4\ -}\ *$

$5 * (3 - 4)$

$(3 - 4) * 5$

Infix notation

$\boxed{3} - \boxed{(4 * 5)}$

operator

operands

Algorithm using Stack: Calculating Postfix Expressions

34 4b 5b * -

? + 25

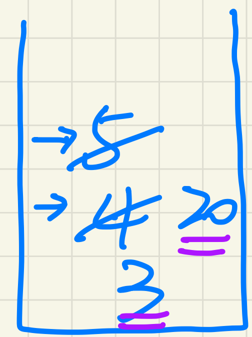
Sketch of Algorithm

- When input is an ***operand*** (i.e., a number), ***push*** it to the stack.
- When input is an ***operator***, obtain its two ***operands*** by ***popping*** off the stack **twice**, evaluate, then ***push*** the result back to stack.
- When finishing reading the input, there should be **only one** number left in the stack.

2 + 3 = 5

5 * 5 = 25

Input 1: 3 4 5 * -	≡ 3 - (4 * 5)
Input 2: 3 4 - 5 *	≡ (3 - 4) * 5
Input 3: 5 2 3 + * +	≡ + 5 * (2 + 3)
Input 4: 5 4 + 6	≡ 5 + 4 6



20

$\boxed{4 * 5}$

LHS RHS

3 - 20

= -17

Lecture

Stack ADT vs. Queue ADT

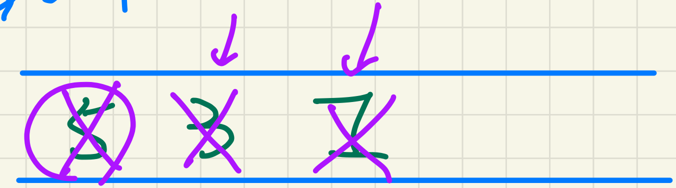
Queue ADT -
First In First Out (FIFO)
Implementations in Java

Queue ADT: Illustration

First-In First-Out

	isEmpty	size	first
new queue	T	0	
enqueue(5)	F	1	5
enqueue(3)	F	2	5
enqueue(1)	F	3	5
dequeue	F	2	3
dequeue	F	1	1
dequeue	T	0	

exception!

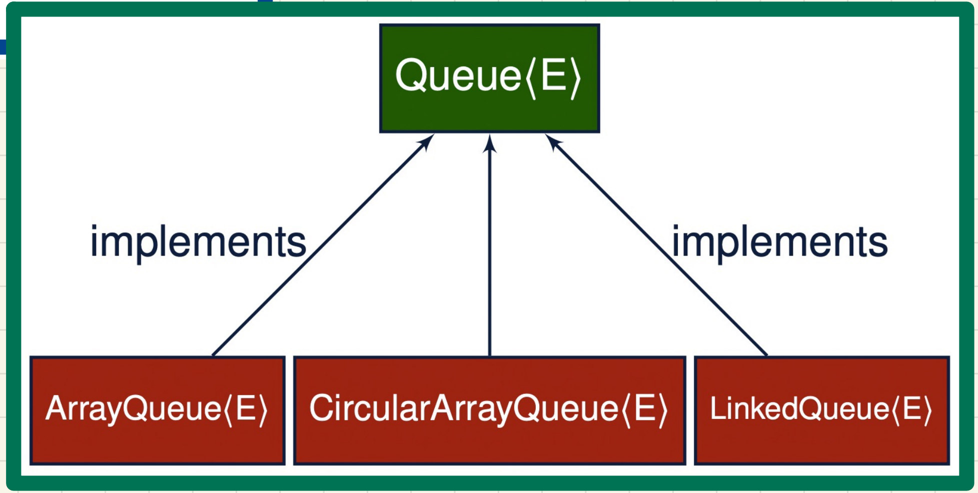


the earlier an element joins a queue, the earlier it gets removed.

exception!

Implementing the Queue ADT in Java: Architecture

```java
public interface Queue< E > {
  public int size();
  public boolean isEmpty();
  public E first();
  public void enqueue( E e);
  public E dequeue();
}
```



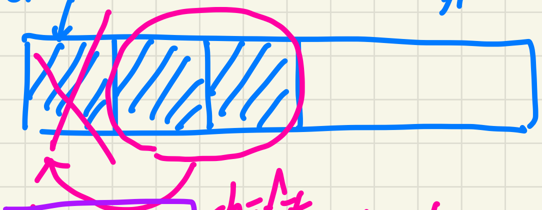
Implementing the Queue ADT using an Array

```java
public class ArrayQueue<E> implements Queue<E> {
  private final int MAX_CAPACITY = 1000;
  private E[] data;
  private int r; /* rear index */
  public ArrayQueue() {
    data = (E[]) new Object[MAX_CAPACITY];
    r = -1;
  }
  public int size() { return (r + 1); }
  public boolean isEmpty() { return (r == -1); }
  public E first() {
    if (isEmpty()) { /* Precondition Violated */ }
    else { return data[0]; }
  }
  public void enqueue(E e) {
    if (size() == MAX_CAPACITY) { /* Precondition Violated */ }
    else { r ++; data[r] = e; }
  }
  public E dequeue() {
    if (isEmpty()) { /* Precondition Violated */ }
    else {
      E result = data[0];
      for (int i = 0; i < r; i ++) { data[i] = data[i + 1]; }
      data[r] = null; r --;
      return result;
    }
  }
}
```

O(1).

O(n)

data (beginning of array → first of Q)



dequeue

O(n)

shift to left by one position

to resolve this → circular array

O(1).