ARRAYCOPY SYNTAX public static void ArrayCopy (source_array, source Pos, dest-arr, dest-pos, len)

JUNIT TESTING assertEquals(expected, actual) assertFalse(<condition) assertArrayEquals assertNotNull(object), assertSame(object expected, Object actual), assertNotSame()

ITERATION INTERFACE

```
Public interface Herable<T>{
  Iterator<T> iterator()
}
public interface Herator<E>
{ boolean hasNext()
  E next()
  Void remove()
  // removes current item}
```

CASTING RULES YOU CAN'T DOWNCAST AN OBJECT UNLESS THE ACTUAL TYPE OF THE OBJECT IS WHAT IT'S BEING CAST TO. CASTING CHANGES THE STATIC TYPE OF AN OBJECT (FOR THE COMPILER'S SAKE)

COMPARISONS THE COMPARABLE INTERFACE ONLY ALLOWS FOR COMPARISON BY ONE ASPECT. COMPARATOR OBJECTS CAN BE MANY, SO FOR EXAMPLE, A "PLAYER" CLASS CAN HAVE RANKCOMPARATOR, SIZE COMPARATOR

INHERITANCE + CONSTRUCTORS SUBCLASSES DO NOT INHERIT CONSTRUCTORS. SUBCLASS CONSTRUCTORS MUST BEGIN WITH A CALL TO A PARENT CONSTRUCTOR BUT THIS IS ALWAYS AUTOMATICALLY DONE (WITH THE NO-ARGS CONSTRUCTOR) THIS IS EXPLICITLY SOLVED BY EXPLICITLY CALLING super.constructor(args).

TYPING STUFF COMPILER ONLY SEES STATIC TYPE. RUNTIME CASTING ERRORS OCCUR WHEN A CAST IS PLAUSIBLE, BUT ACTUALLY IMPROPER. STATIC CLASSES MUST BE ABLE TO EXECUTE THE EXACT METHOD SIGN-ATURE, OTHERWISE, COMPILER ERROR! COMPILER SEES STATIC TYPES, RUNTIME SEES ACTUAL (DYNAMIC) TYPES. FOR ARGUMENTS, WE LOOK AT STATIC TYPES. DYNAMIC TYPE OF ARGUMENTS DOESN'T MATTER.

STATIC METHODS STATIC METHODS ARE NOT OVERRIDDEN. THEY ARE ONLY SELECTED BY LOOKING AT THE STATIC CLASS/OBJECT OF THE CALLER.

STACK MANIPULATION EXAMPLE

```
default void purge(Item x) {
if (size == 0) {return;}
Item top = pop()
purge(x) // remaining
if (!x.equals(top)) {
    push(top)
}
```

INPLACE REVERSE SLL

```
Node prev = null
Node temp = head
while (temp != null) {
  Node next = temp.next
  temp.next = prev
  prev = temp
  temp = next //OG next}
```

NON DESTRUCTIVE REV(SLL)

1) CREATE NEW EMPTY LIST
2) MAKE POINTER VARIABLE FOR THE NEW EMPTY LIST
3) ITERATE (OR TAIL RECURSIVE) THROUGH BOTH LISTS UNTIL ptr == null
4) RETURN HEAD POINTER TO THE NEW LINKED LIST

EXCEPTION SYNTAX

```
// SOME CODE...
throw new Exception()
// ALTERNATIVELY
try{ Some code...
catch{ ExceptionType
Print(something)}
```

VARIOUS REMINDERS [AND NAGS]

* ALWAYS DO A STATIC CHECK!
* FUNCTIONS VS. VARIABLE ACCESS
* NO ARGUMENT CHILD CONSTRUCTORS CALL super() ALWAYS
* GENERIC ARRAY INSTANTIATION IS T[] demBoiz = (T) new Object[8]
* USE API METHODS INSTEAD OF WRITING YOUR OWN IF POSSIBLE
* LINKEDLISTDEQUE(), NOT DEQUE()

INTERFACE RULES * SEMICOLONS IF NOT PROVIDED YOU CAN NEVER DIRECTLY INSTANTIATE A NEW OBJECT AS AN INTERFACE, BUT INTERFACES CAN BE ON THE LEFT SIDE DURING INSTANTIATION. ex InterfaceV a = new ImplementorClass();

HIGHER ORDER FUNCTIONS TAKE ADVANTAGE OF IT.

INSTANTIATING A NESTED CLASS
PRIVATE NESTED CLASSES CAN BE INSTANTIATED IN THE OUTER CLASS.
PUBLIC NESTED CLASSES CAN BE INSTANTIATED FROM ANYWHERE
IF IT IS NONSTATIC A SPECIFIC INSTANCE MUST BE MADE

CALLING OWN CONSTRUCTOR ANOTHER CONSTRUCTOR OF THE SAME CLASS USING this() OR this(PARAMETERS)

PRIANKA SUBRAHMANYAM
CS 61B SPRING 2019 MT1