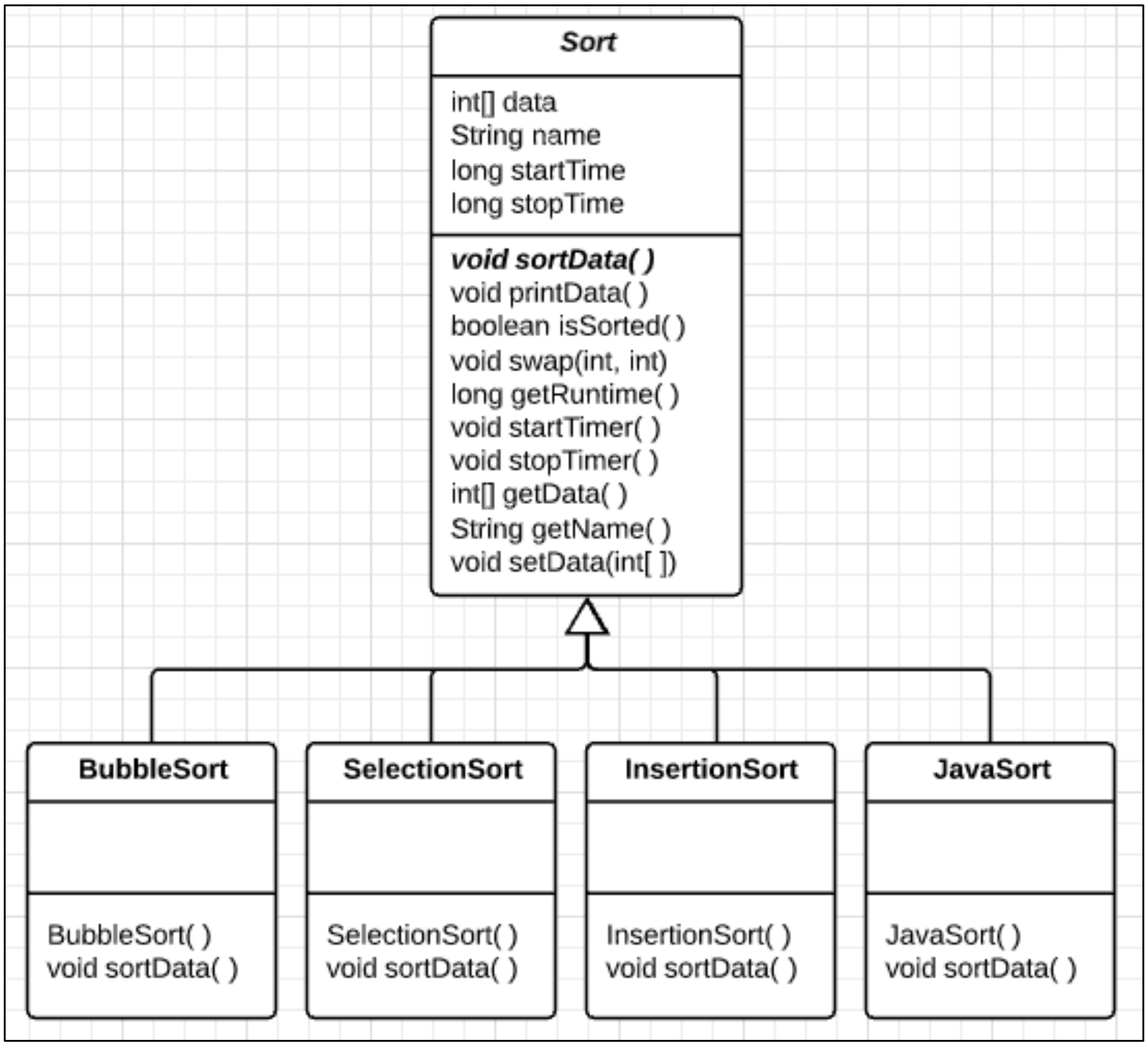
<u>Homework 03: Sort Strategy</u>
CS 412

For this homework, you must implement a version of the Strategy Design Pattern using various Sort algorithms. Specifically, you must have **one** *abstract* Sort class and **four** *concrete* Sort subclasses that inherit and extend Sort's behavior. Your code must implement the class hierarchy defined in the following UML diagram. Additionally, your code MUST work with the provided Main class and main method.



How to interpret the UML diagram:

- *Abstract* class **Sort**
    - Instance variables:

int[] data:	data to be sorted
String name:	name of algorithm
long startTime:	store algorithm start time (in nanoseconds)
long stopTime:	store algorithm stop time (in nanoseconds)

-
    - Methods:

void sortData( )	***abstract* method; subclasses must implement**
void printData( )	prints data to console; one per line (for debugging)
boolean isSorted( )	returns true if data is sorted, false otherwise
void swap(int i, int j)	swaps data between indices
long getRuntime( )	returns difference b/t stop and start time
void startTimer( )	sets startTime to current time (in nanoseconds)
void stopTimer( )	sets stopTime to current time (in nanoseconds)
int[] getData( )	returns the data array
String getName( )	returns the algorithm's name
void setData(int [])	sets the data array instance variable

- Concrete class **BubbleSort**
    - *Inherits* from abstract Sort class
    - BubbleSort( ) constructor: sets the name instance variable to "BubbleSort"
    - Implements concrete sortData( ) method
        - Sorts data array via bubble sort algorithm
        - Calls startTimer( ) as first line in method
        - Calls stopTimer( ) as last line in method

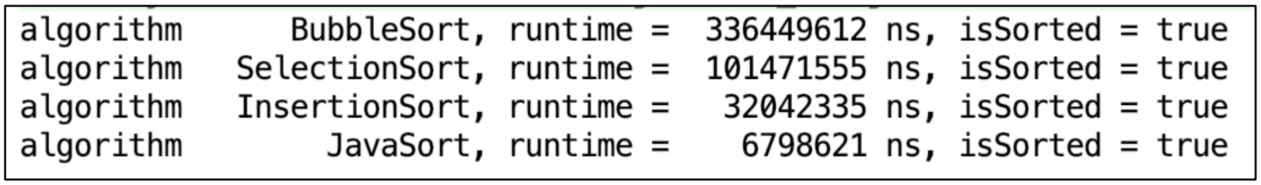
- Concrete class **SelectionSort**
    - *Inherits* from abstract Sort class
    - SelectionsSort( ) constructor: sets the name instance variable to "SelectionSort"
    - Implements concrete sortData( ) method
        - Sorts data array via selection sort algorithm
        - Calls startTimer( ) as first line in method
        - Calls stopTimer( ) as last line in method

- Concrete class **InsertionSort**
    - *Inherits* from abstract Sort class
    - InsertionSort( ) constructor: sets the name instance variable to "InsertionSort"
    - Implements concrete sortData( ) method
        - Sorts data array via insertion sort algorithm
        - Calls startTimer( ) as first line in method
        - Calls stopTimer( ) as last line in method

- Concrete class **JavaSort**
    - Inherits from abstract Sort class
    - JavaSort( ) constructor: sets the name instance variable to "JavaSort"
    - Implements concrete sortData( ) method
        - Sorts data array via Java's built-in sort functionality
        - Calls startTimer( ) as first line in method
        - Calls stopTimer( ) as last line in method

EXAMPLE:

- Sample output:



```
algorithm      BubbleSort, runtime =  336449612 ns, isSorted = true
algorithm   SelectionSort, runtime =  101471555 ns, isSorted = true
algorithm   InsertionSort, runtime =   32042335 ns, isSorted = true
algorithm        JavaSort, runtime =    6798621 ns, isSorted = true
```

HINTS:

- abstract keyword
- extends
- public or protected
- long timeInNano = System.nanoTime( );
- Arrays.sort(array);
- isSorted( ):
    - for each element, if next element is greater than current, array is NOT sorted..

Sort algorithm pseudocode:

- BubbleSort

```
bubbleSort(array):
    N = size(array)
    swapped = true
    while swapped:
        swapped = false
        for i = 0; i < N-1; i++:
            if array[i] > array[i+1]:
                swap(array[i], array[i+1])
                swapped = true
```

- SelectionSort

```
selectionSort(array):
    N = size(array)
    for i = 0; i < N-1; i++:
        index = i
        for j = i + 1; j < N; j++:
            if array[j] < array[index]:
                index = j
        if index != i:
            swap(array[i], array[index]
```

- InsertionSort

```
insertionSort(array):
    N = size(array)
    for i = 1; i < N; i++:
        val = array[i]
        loc = i – 1
        while(loc >= 0 && array[loc] > val):
            array[loc+1] = array[loc]
            loc -= 1
        array[loc+1] = val
```