

SE102 Abstract Data Type and Problem Solving

Assist Prof. Pree Thiengburanathum, PhD.

Announcement

- Comp Org Module overlapping

Agenda

- Global and local variable
- Control flow
- Dynamic array
- Basic searching and sorting

Object and class

- A class is a blueprint
- An object is an instance created from that blueprint
- All objects of the same class have the same set of attributes
 - Every person object have name, weight, height
- But different value for those attributes
 - p.name = pree, p.name = john

Structure of a Java class

- Blue print of an object
- Contains attributes and behaviors
- At run time, it will create instance of the object

Syntax of class:

```
class classname
{
    type instance-variable;
    type methodname1(parameter-list)
    {
        // body of method
    }
    type methodname2(parameter-list)
    {
        // body of method
    }
}
```

Java basics- method

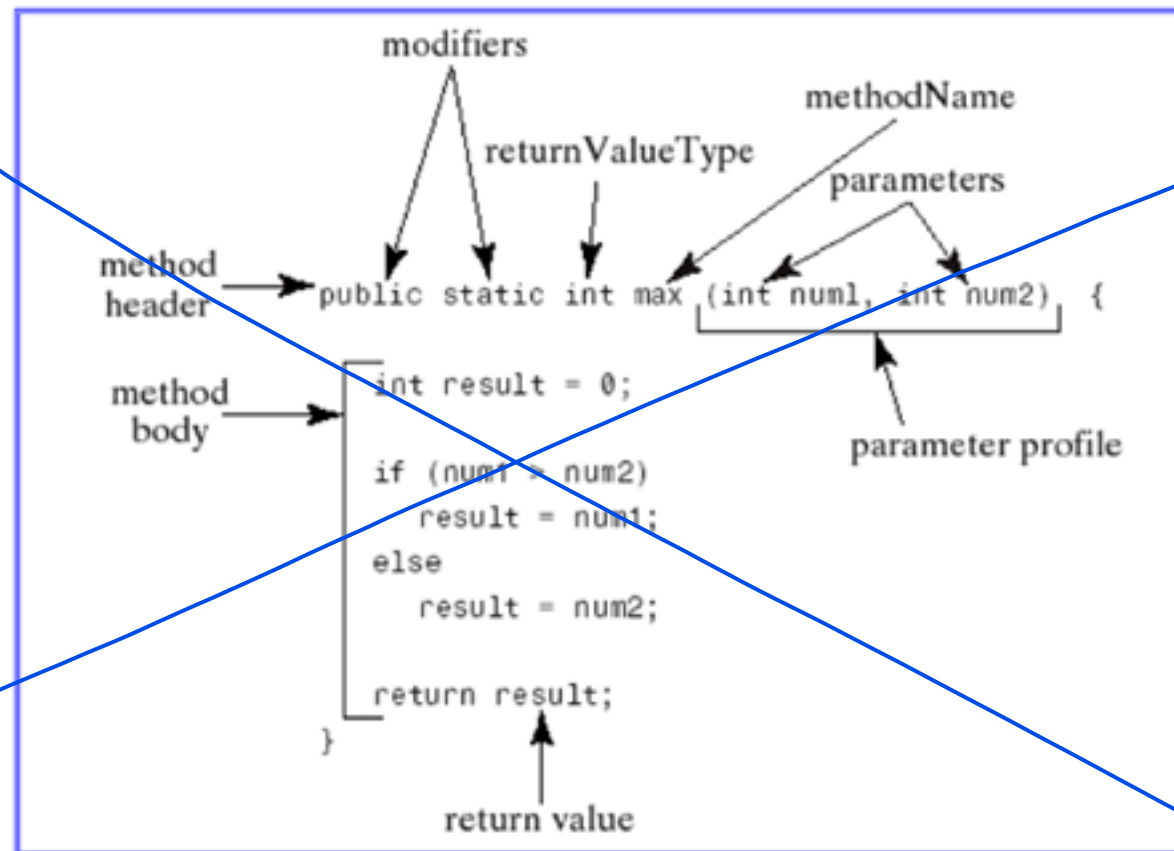
- **Parameters**

- Parameter list may be empty
- Parameter list consists of comma separated pairs of types and parameter names.

```
Public void setAge(String name, int age) {  
}
```

Java basics- method

Method Structure



Java basics- methods

- **Constructors**
- Used to initialize new objects
- Has the same name as class and no return type

```
public counter() {  
    count =0;  
}
```

```
Public Professor (String name, String dept) {  
    name = name;  
    dept = dept;  
}
```


Java basics- methods

- Block and Local variables
- Body of method is called block
 - A sequence of statements and declarations enclosed in branches ({});
 - Blocks may have blocks nested
 - Variables declared with a block are known only in that block
 - These variable are called local variables

```
public static int sumThree(int a, int b, int c){  
    int sum;  
    int partsum = a + b;  
    sum = partsum + c;  
    return sum;  
}
```

Java basic – control flow

If Statement

- `if (boolean_exp) {
 what_to_do_if_true
}`
- `if (boolean_exp) {
 what_to_do_if_true
}
else {
 what_to_do_if_false
}`
- `if (1st_boolean_exp) {
 what_to_do_if_1st_true
}
else if (2nd_boolean_exp) {
 what_to_do_if_2nd_true
}
else {
 what_to_do_if_all_false
}`

Java basic – control flow

Switch Statement

```
□ switch (int_type_exp) {  
    case CONST1:  
        action_for_CONST1;  
        break;  
    case CONST1:  
        action_for_CONST1;  
        break;  
    case CONST2:  
        action_for_CONST2;  
        break;  
    case CONST3:  
        action_for_CONST3;  
        break;  
    . . .  
    default:  
        action_for_no_match;  
        break;  
}
```

Java basic – control flow

Switch Statement Example

```
□ switch (stars) {  
    case 4:  
        message = "truly exceptional";  
        break;  
    case 3:  
        message = "quite good";  
        break;  
    case 2:  
        message = "fair";  
        break;  
    case 1:  
    case 0:  
        message = "forget it";  
        break;  
    default:  
        message = "no info found";  
        break;  
}
```

Java basic – control flow

While Loops

□ Syntax

```
initialize
while (boolean_exp) {
    work_to_be_done
    update
}
```

□ Example

```
int counter = 10;
while (counter > 0) {
    System.out.println(counter);
    counter--;
}
System.out.println("Blast Off!");
```

□ What is the output?

□ What if we exchange order of two statements in loop?

Java basic – control flow

For Loops

- Syntax

```
for (initialization; boolean_exp; update) {  
    work_to_be_done  
}
```

- Example

```
for (int counter = 10; counter > 0; counter--) {  
    System.out.println(counter);  
}  
System.out.println("Blast Off!");
```

- What is the output?
- When is update performed?
- What is value of counter after loop?

Java basic – control flow

Do-While Loops

□ Syntax

```
initialize  
do  
{  
    work_to_be_done  
    update  
} while (boolean_exp);  
○ NOTE REQUIRED SEMICOLON!!!
```

□ Example

```
int counter = 10;  
do {  
    System.out.println(counter);  
    counter-- ;  
} while (counter > 0);  
System.out.println("Blast Off!");
```

Java basics - loops

- Which kind of loop do we use?
- While loop
 - Don't know of often its going to be
 - Update could be anywhere in the loop
- For loop
 - Know how often in advance
 - All information controlling loop together, in front

Tokenize

- Break a string into tokens
- Java has a class which is called “StringTokenizer”
- The StringTokenizer class helps splitting Strings in to multiple tokens.

Sample data file

10,2,4,7

3,3,1

1,3,7

2

StringTokenizer (cont.)

String []

```
public List<String> getTokens(String str) {  
    List<String> tokens = new ArrayList<>();  
    StringTokenizer tokenizer = new StringTokenizer(str, ",");  
    while (tokenizer.hasMoreElements()) {  
        tokens.add(tokenizer.nextToken());  
    }  
    return tokens;  
}
```

Tokenize

- Break a string into tokens
- Java has a class which is called “StringTokenizer”
- The StringTokenizer class helps splitting Strings in to multiple tokens.

Sample data file

10,2,4,7

3,3,1

1,3,7

2

StringTokenizer (cont.)

```
public List<String> getTokens(String str) {  
    List<String> tokens = new ArrayList<>();  
    StringTokenizer tokenizer = new StringTokenizer(str, ",");  
    while (tokenizer.hasMoreElements()) {  
        tokens.add(tokenizer.nextToken());  
    }  
    return tokens;  
}
```

SE102 ADT Lecture 3

Dynamic array, Basic Searching and Sorting

Pree Thiengburanathum

Vectors

- Vector implements a *dynamic array*.
- An important difference between an array and a vector is that a vector can be thought of as a dynamic, able to *change its size as needed*.
- If an element is added that doesn't fit in the existing space, more room is automatically acquired.
- Vectors can hold objects of any type and any number.
- Vector class is contained in *java.util.package*

Declaring Vectors

vector<String> v = new vector<String>(11)

- `Vector a = new Vector();` -> Creates a default vector, which has an initial size of 10.
- `Vector a = new Vector(int size);` -> Creates a vector, whose initial capacity is specified by size.
- `Vector a = new Vector(int size, int incr);` -> Creates a vector, whose initial capacity is specified by size and whose increment is specified by incr.

Class Vector constructors

- **public Vector();**
- Constructs an **empty vector** so that its **internal data array has size 10.**
- e.g. `Vector v = new Vector();`
- **public Vector(int initialCapacity);**
- Constructs an empty vector with the **specified initial capacity.**
- e.g. `Vector v = new Vector(1);`
- **public Vector(int initialCapacity, int capacityIncrement);**
- Constructs an empty vector with the specified initial capacity and capacity increment.
- e.g. `Vector v = new Vector(4, 2);`

Class **Vector** methods

public void addElement(Object obj)

Adds the specified component to the end of this vector, increasing its size by one. The capacity of this vector is increased if its size becomes greater than its capacity.

e.g. `v.addElement(input.getText());`

public boolean removeElement(Object obj)

Removes the first (lowest-indexed) occurrence of the argument from this vector. If the object is found in this vector, each component in the vector with an index greater or equal to the object's index is shifted downward to have an index one smaller than the value it had previously.

e.g. `if (v.removeElement(input.getText()))
 showStatus("Removed: " + input.getText());`

Class **Vector** methods

public Object firstElement()

Returns the first component (the item at index 0) of this vector.

e.g. showStatus(**v.firstElement()**);

public Object lastElement()

Returns the last component of the vector.

e.g. showStatus(**v.lastElement()**);

public boolean isEmpty()

Tests if this vector has no components.

e.g. showStatus(**v.isEmpty()**? “Vector is empty” : “Vector is not empty”);

Class **Vector** methods

public boolean contains(Object elem)

Tests if the specified object is a component in this vector.

e.g. if (**v.contains(input.getText())**)

showStatus("Vector contains: " + input.getText());

public int indexOf(Object elem)

Searches for the first occurrence of the given argument, testing for equality using the equals method.

e.g. **showStatus**("Element is at location" + **v.indexOf(input.getText())**);

public int size()

Returns the number of components in this vector.

e.g. **showStatus**("Size is " + **v.size()**);

Class **Vector** methods

public boolean contains(Object elem)

Tests if the specified object is a component in this vector.

e.g. if (**v.contains(input.getText())**)

showStatus("Vector contains: " + input.getText());

public int indexOf(Object elem)

Searches for the first occurrence of the given argument, testing for equality using the equals method.

e.g. showStatus("Element is at location" + **v.indexOf(input.getText())**);

public int size()

Returns the number of components in this vector.

e.g. showStatus("Size is " + **v.size()**);

Given a array, Search a given element in array.

Case 1:

Input: Search 20.

12	5	10	15	31	20	25	2	40
0	1	2	3	4	5	6	7	8

Output: True (20 is present in array)

Case 2:

Input: Search 26.

12	5	10	15	31	20	25	2	40
0	1	2	3	4	5	6	7	8

Output: False (26 is not present in array)

Workshop

- Implement the linear search method which takes two parameters (i.e. array of integer and value to search)
- Call the method from the main

Solution

```
1 public class LinearSearch {  
2     public static void main(String a[]){  
3         int[] arr = {12, 5, 10, 15, 31, 20, 25, 2, 40};  
4         int dataToSearch = 40;  
5  
6         boolean isFound = search(arr, dataToSearch);  
7         if(isFound){  
8             System.out.println("Match found");  
9         }else{  
10            System.out.println("Match Not found");  
11        }  
12    }  
13  
14    public static boolean search(int[] arr, int dataToSearch){  
15        for(int i=0; i<arr.length; i++){  
16            if(arr[i] == dataToSearch){  
17                return true;  
18            }  
19        }  
20        return false;  
21    }  
22 }  
23  
24  
25  
26 }
```

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Binary Search

$$\frac{0 + 6}{2} = 3$$

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

$$\frac{0 + 3}{2}$$

Find approximate midpoint

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

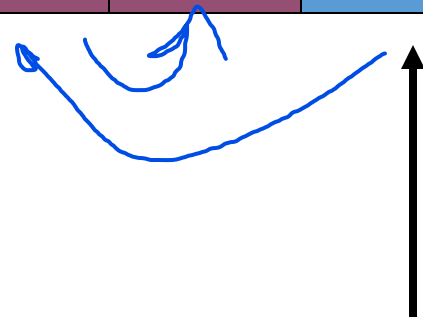


Is 7 = midpoint key? NO.

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53



Is 7 < midpoint key? YES.

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53



Find approximate midpoint

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53



Target = key of midpoint? NO.

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53



Target < key of midpoint? NO.

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

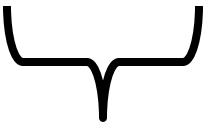


Target > key of midpoint? YES.

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53



Search for the target in the area after midpoint.

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53



Find approximate midpoint.
Is target = midpoint key? YES.

Sorting

- Arranging in an ordered sequence is called "sorting". Sorting is a common operation in many applications, and efficient algorithms to perform it have been developed.



Common sorting algorithms

- **Bubble sort**- exchange two adjacent elements if they are out of order. Repeat until array is sorted
- **Insertion sort**- scan successive elements for an out-of-order item, then insert the item in the proper place.
- **Selection sort**- find the smallest element in the array, and put it in the proper place. Swap it with the value in the first position. Repeat until array is sorted.

Common sorting algorithms

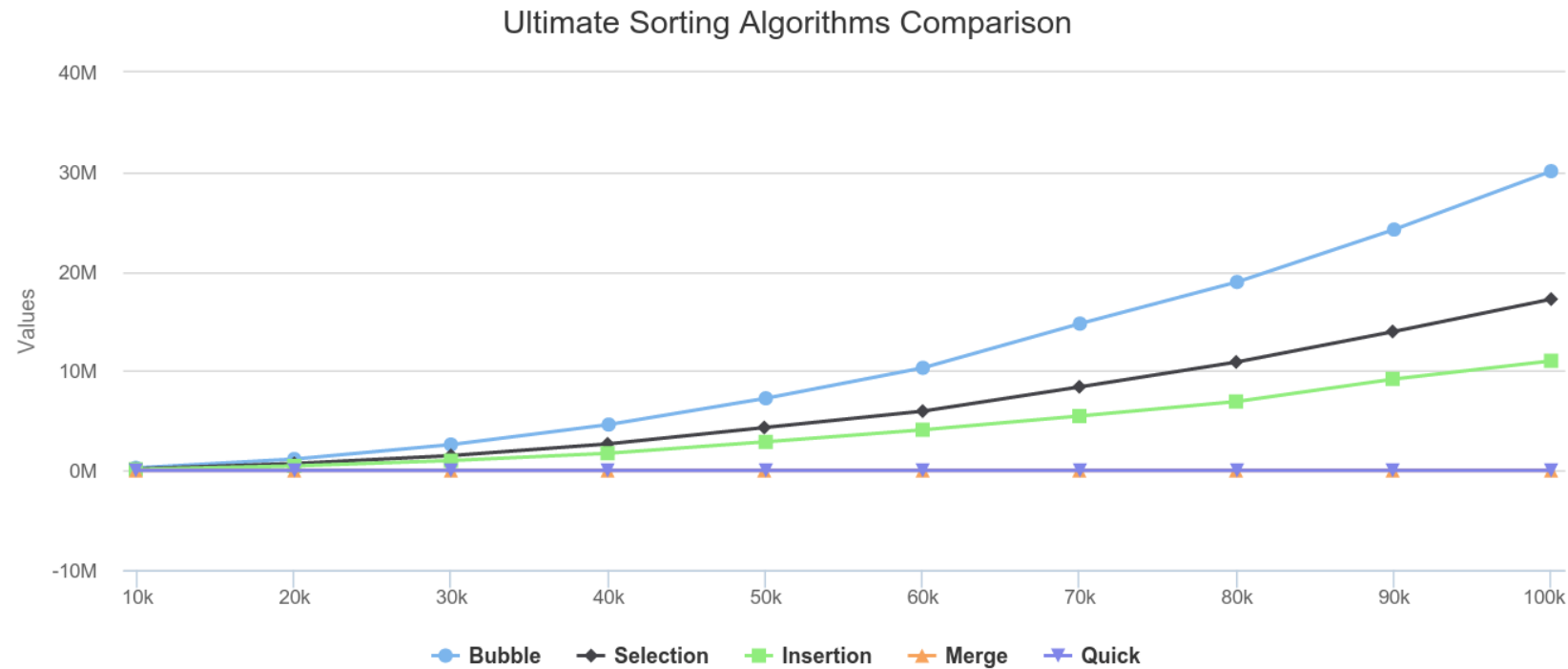
- Quick sort

Partition the array into two segments. In the first segment, all elements are less than or equal to the pivot value. In the second segment, all elements are greater than or equal to the pivot value. Finally, sort the two segments recursively.

- Merge sort

Divide the list of elements in two parts, sort the two parts individually and then merge it.

Sorting Algorithms Comparison



Selection Sort

12	12	22	14	8	17
6	12	22	14	12	17

- Given n numbers to sort:
- Repeat the following n-1 times:
 - Mark the first unsorted number
 - Find the smallest unsorted number
 - Swap the marked and smallest numbers

Selection Sort

6	8	22	14	12	17
6	8	12	14	22	22

- Given n numbers to sort:
- Repeat the following n-1 times:
 - Mark the **first** unsorted number
 - Find the **smallest** unsorted number
 - Swap the **marked** and **smallest** numbers

Selection Sort

- Given n numbers to sort:
- Repeat the following $n-1$ times:
 - Mark the first unsorted number
 - Find the smallest unsorted number
 - Swap the marked and smallest numbers

3 steps

- How efficient is selection sort?
 - In general, given n numbers to sort, it performs n^2 comparisons
- Why might selection sort be a good choice?
 - Simple to write code
 - Intuitive

Selection Sort

- Given n numbers to sort:
- Repeat the following $n-1$ times:
 - Mark the first unsorted number
 - Find the smallest unsorted number
 - Swap the marked and smallest numbers

Try one!

15	3	11	19	4	7
----	---	----	----	---	---

Class exercise:

Write a java application program that uses **Vector** class and can output the result as shown in the right.

Output:

Enter lines of input, use quit to end the program.

apple
orange
banana
grape
lemon
quit

Number of lines: 5

First line: apple

Lines in reverse order:

lemon
grape
banana
orange
apple

Questions?

Question?