Faculty of Science and Engineering

COMP125 Fundamentals of Computer Science
Workshop - Searching and Arrays of Objects

Learning outcomes

By the end of this session, you will know the basics of objects and testing in Java. In particular, you will be able to,

a. Perform binary search on an array;

b. Operate on array of objects;

c. Create array of objects

Import project `searchingWorkshopTemplate` from `searchingWorkshopTemplate.zip`.

1. **Apply binary search on an array**

   Consider the following **pseudo-code** is being used to perform binary search.

```
Input:
1. array arr: assumed to be sorted in ascending order
2. target (of the type as each item of the array arrr)

Output:
index in array at which target is found. -1 if not found

Process:
//Search space: from first to last item
first = 0
last = arr.length - 1
while(first <= last)
   median = (first+last)/2
   if target is equal to arr[median]
      OUTPUT median
   //options left target < arr[median] or target > arr[median]
   if target < arr[median] //if present, it's in the left half
      last = median - 1
   else //target > arr[median] means if present, it's in the right
      half
      first = median + 1
end while loop

//loop ending implies first > last and search space exhausted
OUTPUT -1
```

For the array

```
{1, 4, 5, 7, 9, 23, 47, 50, 58, 58, 58, 58, 88, 90, 95, 99}
```

Trace the execution of the above algorithm for the following targets,

- 58

first	last	median	arr[median]

- 50

first	last	median	arr[median]

- 47

first	last	median	arr[median]

- 59

first	last	median	arr[median]

2. **Variations of binary search**

Write down, on a piece of paper, the changes you need to make to the binary search code provided in class `ArrayService`,

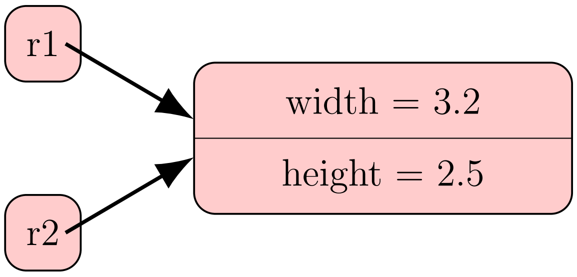
a. to search for an item in an integer array sorted in descending order

> **Solution:**
>
> `if(target > arr[median])`
>
> `is replaced by`
>
> `if(target < arr[median])`
>
> Similarly,
>
> `if(target < arr[median])`
>
> `is replaced by`
>
> `if(target > arr[median])`

b. to search for a `Rectangle` object in an array of `Rectangle` objects.
You cannot compare two objects, `r1, r2` of a class `Rectangle`, for equality as `if(r1 == r2)`.
The primitive equality operator (`==`) checks if `r1` and `r2` refer to the same object (memory block).
Or in other words, `r1` and `r2` are shallow copies of each other.
Consider the following code:

```java
Rectangle r1 = new Rectangle(3.2, 2.5);
Rectangle r2 = r1;
```

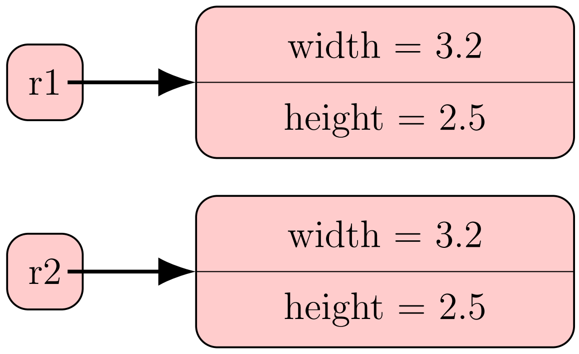
Here, `r2` is said to be a *shallow copy* of `r1`. That is, we are copying the reference `r1` into `r2`. Here, `r1 == r2` would evaluate to `true`. The corresponding memory diagram is below:



Now, consider the following code:

```java
Rectangle r1 = new Rectangle(3.2, 2.5);
double w = r1.getWidth();
double h = r1.getHeight();
Rectangle r2 = new Rectangle(w, h);
```

Here, `r2` is said to be a *deep copy* of `r1`. We are, in effect, *cloning* the object `r1` into `r2`. In this case, `r1 == r2` would evaluate to `false`. The corresponding memory diagram is below:



Instead, we must use `if(r1.compareTo(r2) == 0)`. This is, of course, assuming the following method is defined in the `Rectangle` class.

```java
public int compareTo(Rectangle other) {
        if(this.area() > other.area())
                return 1;
        if(this.area() < other.area())
                return -1;
        return 0;
}
```

The following summarises comparison of primitive data type variables vs. comparison of objects.

Comparison of primitive data type variables a and b	Corresponding comparison of objects `obj1` and `obj2`
`a == b`	`obj1.compareTo(obj2) == 0`
`a != b`	`obj1.compareTo(obj2) != 0`
`a > b`	`obj1.compareTo(obj2) == 1`
`a < b`	`obj1.compareTo(obj2) == -1`
a $\geq$ b	`obj1.compareTo(obj2) != -1`
a $\leq$ b	`obj1.compareTo(obj2) != 1`

Solution:

`if(target == arr[median])`

`is replaced by`

`if(target.compareTo(arr[median]) == 0)`

Similarly,

`if(target < arr[median])`

`is replaced by`

`if(target.compareTo(arr[median]) < 0)`

and also,

`if(target > arr[median])`

`is replaced by`

`if(target.compareTo(arr[median]) > 0)`

3. **Understanding the process of creating array of objects**

   In pairs, discuss the contents of `arrayOfObjectsSummary.pdf` (included in the `eclipse project` that you imported) and repeat the process illustrated in it for the class `Trip` and the client `TripListClient` that creates an array of `Trip` objects and operates on it.

4. **Create an array of objects**

   You are provided with a completed `StockItem` class. Complete the code in `StockItemListClient` that creates and operates on an array of `StockItem` objects, similar to the code in `TripListClient` that creates and operates on an array of `Trip` objects. You can use the following configuration values,

   - array should be of size 5
   - `name` of first item should be "Item 1", of second item should be "Item 2" and so on. In general, `name` of item at index `i` should be `"Item "+(i+1)`.
   - `unitPrice` of item at index `i` should be `2*i + 2`. (you can assign an integer to a double variable so it's all good)
   - `quantity` of item at index `i` should be `2*i - 8`.

   Display the item with the maximum `totalStockPrice()` as defined in the `StockItem` class.

5. **Additional Tasks (time permitting)**

Complete the following methods based on the specifications,

a. **existsInFirstHalf**: returns `true` if `target` exists in the first half of the array `arr`, and `false` otherwise. If the array contains an odd number of items ($2 * k + 1$, where k is an integer), then the first half contains the first k items.

```java
public static boolean existsInFirstHalf(double[] arr, double
    target);
```

```java
                public static boolean existsInFirstHalf(double[]
                    arr, double target) {
                        if(arr == null)
                                return false;
                        int end = arr.length/2;
                        for(int i=0; i < end; i++) {
                                if(arr[i] == target) {
                                        return true;
                                }
                        }
                        return false;
                }
```

b. **getCount**: returns the number of times `target` exists in `arr`.

```java
public static int getCount(double[] arr, double target);
```

```java
                public static int getCount(double[] arr, double
                    target) {
                        if(arr == null)
                                return 0;
                        int count = 0;
                        for(int i=0; i < arr.length; i++) {
                                if(arr[i] == target) {
                                        count++;
                                }
                        }
                        return count;
                }
```

The time complexity in the best case is $\mathcal{O}(log_2(n))$ and in the worst or average case is $\mathcal{O}((n)$.

c. **getMostFrequentItem**: Assuming the method `getCount` is completed and available, returns the item that occurs the most number of times in `arr`. If there is a tie, the candidate that occurs first in the array is returned. For example, if $arr = \{8.5, 8.5, 8.5, 1.7, 1.7, 1.7, 6.2, 6.2, 6.2\}$, the method returns `8.5`. Return 0 if the array is empty or null.

```java
public static double getMostFrequentItem(double[] arr);
```

```java
public static double getMostFrequentItem(double[] arr) {
        if(arr == null || arr.length == 0)
                return 0;
        int freqIndex = 0;
        for(int i=1; i < arr.length; i++) {
                if(getCount(arr, arr[i]) > getCount(arr, arr[
                    freqIndex)) {
                        freqIndex = i;
```

```
            }
        }
        return arr[freqIndex];
}
```

d. **getFirstIndex**: Consider the following method for binary search.

```
public static int binarySearch(double[] arr, double target) {
        int first = 0;
        int last = arr.length - 1;
        while(first <= last) {
                int median = (first+last)/2;
                if(target==arr[median])
                        return median;
                if(target > arr[median])
                        first = median + 1;
                else
                   last = median - 1;
        }
        return -1;
}
```

The above method returns index 3 while searching for the item 24 in the array $\{0, 24, 24, 24, 24, 24, 24\}$. Modify the method so that it returns the **first** index of the item from the array (should return 1 for the above example).

```
public static int binarySearchV2(double[] arr, double target) {
        int first = 0;
        int last = arr.length - 1;
        int median = -1;
        while(first <= last) {
                median = (first+last)/2;

                if(target == arr[median])
                        break; //exit the loop

                if(target > arr[median])
                        first = median + 1;
                else
                   last = median - 1;
        }
        if(median == -1)
                return -1;

        while(median >= 0 && target == arr[median]) {
                median--;
        }

        return median + 1;
}
```

e. Write a method in a Client class that when passed an array `arr` of `Rectangle` objects, returns an array with **deep copies** of `Rectangle` objects from `arr` that are not `null`. Return `null` if `arr` itself is null. Return an empty array (not `null`) if `arr` is not `null` but every item of `arr` is `null`.

Solution:

```java
public static Rectangle[] nonNullItems(Rectangle[] arr) {
        if(arr == null)
                return null;

        int count = 0;
        for(int i=0; i < arr.length; i++)
                if(arr[i] != null)
                        count++;

        Rectangle[] result = new Rectangle[count];
        int targetIndex = 0;
        for(int i=0; i < arr.length; i++) {
                if(arr[i] != null) {
                        result[targetIndex] = new Rectangle(arr[i]); //call
                            to copy constructor
                        /* DO NOT USE
                        result[targetIndex] - arr[i];
                        That is a shallow copy
                        */
                        targetIndex++;
                }
        }
        return result;
}
```