Activity 6-1

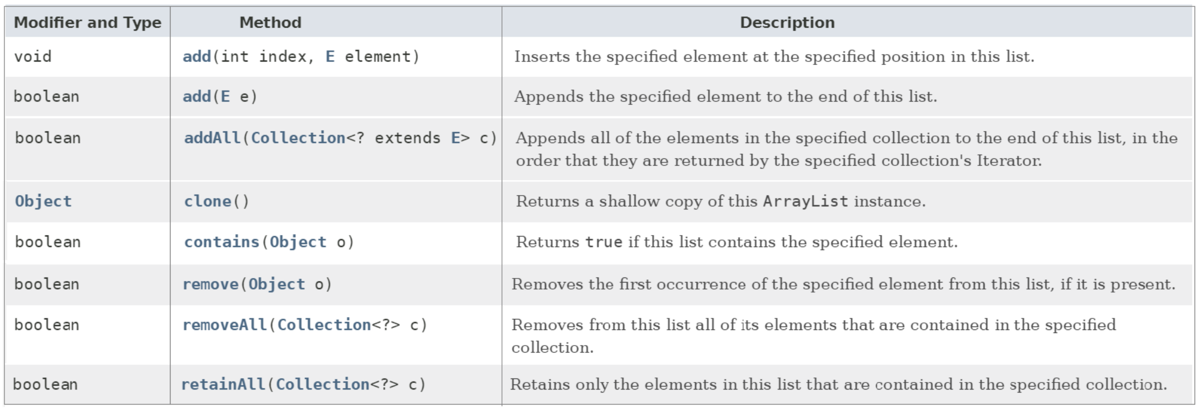
Offshore Islands

1. Create an empty ArrayList, called ***a1***, to store String type.
2. Use ***add() t***o add the following names into the ***a1***.
   * Sentosa
   * Seletar Island
   * Sisters Island
   * St John Island
   * Pulau Ubin

2. Use ***System.out.println (a1)*** to display the content of ***a1***.
3. Remove the element at index ***2*** from ***a1***using ***remove()***.
4. Use ***System.out.println (a1)*** to display the content of ***a1*** again.
5. Edit your code to check if the element to be removed (that specified in step 3) is not found in ***a1***.
6. Use ***contains()*** to check if a given element exists in the ***a1***.
7. Use ***clone()*** to create a 2nd ArrayList, called ***a2***, by cloning ***a1***.
   (For, both ***a1*** and ***a2*** contain identical elements.)
8. Use ***System.out.println (a2)*** to display the content of ***a2***.
9. Use ***add()*** to add a new element (say "***Pulau Tekong***") at index ***2*** for ***a2***.

10. For the ease of experimenting other methods in ArrayList in subsequent steps, create 2 new ArrayList, ***a3*** (by cloning ***a1***) and ***a4*** (by cloning ***a2***).
    (Now, ***a3*** and ***a4*** is a copy of ***a1*** and ***a2*** respectively)
11. Experiment with ***a3.addAll(a4)***. Verify the content of ***a3***.
12. Revert the ***a3*** by cloning from ***a1*** again.
13. Experiment with ***a3.retainAll(a4)***. Verify the content of ***a3***.
14. Revert the ***a3*** by cloning from ***a1*** again.
15. Experiment with ***a4.removeAll(a3)***. Verify the content of ***a4***.
16. What are the functions of ***addAll(), retainAll()*** and ***removeAll()***?

Useful API documentation for class ArrayList:



Modifier and Type	Method	Description
`void`	`add(int index, E element)`	Inserts the specified element at the specified position in this list.
`boolean`	`add(E e)`	Appends the specified element to the end of this list.
`boolean`	`addAll(Collection<? extends E> c)`	Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
`Object`	`clone()`	Returns a shallow copy of this `ArrayList` instance.
`boolean`	`contains(Object o)`	Returns `true` if this list contains the specified element.
`boolean`	`remove(Object o)`	Removes the first occurrence of the specified element from this list, if it is present.
`boolean`	`removeAll(Collection<?> c)`	Removes from this list all of its elements that are contained in the specified collection.
`boolean`	`retainAll(Collection<?> c)`	Retains only the elements in this list that are contained in the specified collection.

Activity 6-2

Mix of Lighthouses and integers

1. Create an ArrayList, called ***a1***, from a string array ***a*** using ***Arrays,asList()***:

```
String a[] = {"Raffles Marina", "Bedok",
               "Sultan Shoal","Fort Canning"};

ArrayList<String> a1 = new ArrayList<>(Arrays.asList(a));
```

2. Create an LinkedList, called ***LL1***, from array ***a1*** using::

```
LinkedList<Object> LL1 = new LinkedList<Object>(a1);
```

3. Use ***removeFirst()*** to remove the first element from ***LL1***.
4. Use ***addFirst()*** to add an ***integer 111*** to the head of ***LL1***.
5. Use ***removeLast()*** to remove the last element from ***LL1***.
6. Use ***addLast()*** to add an ***integer 888*** to the tail of ***LL1***.
7. Use ***ListIterator*** to display the ***LL1*** forward.
8. Use ***ListIterator*** to display the ***LL1*** backward.

Useful API documentation for class LinkedList:

Modifier and Type	Method	Description
`void`	`addFirst(E e)`	Inserts the specified element at the beginning of this list.
`void`	`addLast(E e)`	Appends the specified element to the end of this list.
`ListIterator<E>`	`listIterator(int index)`	Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
`ListIterator<E>`	`listIterator()`	Returns a list-iterator of the elements in this list (in proper sequence), starting at the beginning of the list.
`E`	`removeFirst()`	Removes and returns the first element from this list.
`E`	`removeLast()`	Removes and returns the last element from this list.

Useful API documentation for class ListIterator:

Modifier and Type	Method	Description
`boolean`	`hasNext()`	Returns `true` if this list iterator has more elements when traversing the list in the forward direction.
`boolean`	`hasPrevious()`	Returns `true` if this list iterator has more elements when traversing the list in the reverse direction.
`E`	`next()`	Returns the next element in the list and advances the cursor position.
`E`	`previous()`	Returns the previous element in the list and moves the cursor position backwards.

Activity 6-3

Reverse A Stack of Books

1. Create a class ***Book*** with this skeleton (you will need to edit it later):

```
class Book {
    String ISBN, author, title;
    public Book(String ISBN, String author, String title) {
        this.ISBN = ISBN;
        this.author = author;
        this.title = title;
    }
}
```

2. In the ***main***(), create a ***Stack***, called ***books***, to hold 3 ***Book*** objects with the information by ***push()*** in the order as shown below::

ISBN	TITLE	AUTHOR
1234	Fundamental of Java	JJ
3456	Fundamental of Kotlin	KK
5678	Fundamental of Python	PP

3. Display the content of stack ***books*** using ***books.forEach(e->System.out.println(e))*** as:
 (you need modify the class ***Book*** to get this output)

```
1234-Fundamental Java-by JJ
3456-Fundamental Kotlin-by KK
5678-Fundamental Python-by PP
```

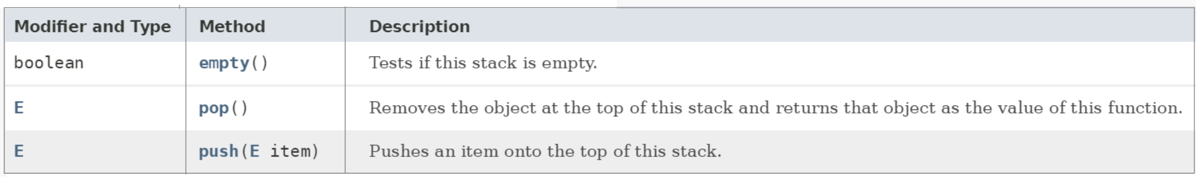
4. Create a 2nd ***Stack*** called ***booksReversed*** to hold ***Book*** objects.
5. Construct a loop to ***pop()*** the topmost element from ***books***, and then ***push()*** it to ***booksReversed*** until ***books*** is empty.

6. Display the content of stack ***books*** using ***books.forEach(e->System.out.println(e))*** as:

```
5678-Fundamental Python-by PP
3456-Fundamental Kotlin-by KK
1234-Fundamental Java-by JJ
```

 (you need to reference the ***books*** properly to get this output)

Useful API documentation for class Stack:



Modifier and Type	Method	Description
boolean	empty()	Tests if this stack is empty.
E	pop()	Removes the object at the top of this stack and returns that object as the value of this function.
E	push(E item)	Pushes an item onto the top of this stack.

Activity 6-4

Internship

1. Assuming that there are 4 tracks of internships: ***IoT***, ***AI***, ***Robotics*** and ***Web***.
2. Students need to fill in their choices, minimum 1 and maximum 4 choices. Assuming that not all students fill up all the choices.
3. Students are ranked by their ***GPA*** in a ***PriorityQueue***, from highest to lowest.
4. For this exercise, you just need to complete task 5 - 8:
5. Create a class Student with ***GPA***, ***adm*** and a ***HashMap*** storing their choices. For instance, the ***HashMap*** of a student who has filled in 3 choices:

1	AI
2	Web
3	IoT

```
class Student {
   String adm;
   double GPA;
   HashMap<Integer, String> choices  ;

   public Student(String adm, double GPA, HashMap<Integer, String> choices) {
        this.adm = adm;
        this.GPA = GPA;
        this.choices = choices;
   }
}
```

6. Create 3 Student objects with appropriate data and varied number of choices.
7. Create a ***PriorityQueue*** to house these 3 ***Student*** objects, sorted according to their ***GPA***, from highest to lowest.

   Note: As comparator deals with ***Double*** (wrapper class), the comparison between two ***Double*** is;

   Double.compare (*Double1, Double2*)

   It returns 0 if equal, negative if ***Double1*** is smaller and positive if ***Double1*** is bigger.

8. Display the ***adm*** and ***GPA*** of Students in the ***PriorityQueue*** using ***forEach***().

Task 9 -14 is Bonus Question for Activity 6 - 4

9. Create another HashMap for storing the vacancies in each of the tracks: IoT, AI, Robotics and Web. Example:

IoT	2
AI	1
Robotics	2
Web	1

10. A student is picked up (one by one) from the head of the queue and assigned with his/her choice 1, if there is still vacancy. Otherwise, the algorithm will go down the list of choices made by the student and check against the vacancies for each of these choices to find a match.
11. If a choice is being assigned to a student, record the assigned track with the Student object (Yes, an additional attribute in class Student is needed to store this) and decrement the vacancy for the respective track by 1.
12. There is a possibility that a student may not be assigned with any track.
13. Display the assigned track (NIL if not assigned) for all students.
14. Display the status of the vacancies for the 4 tracks after the assignment process.

Activity 6-5

CCA

1. Create a ***TreeSet***, called ***ts1***, to store String.
2. Use add() to add ***netball***, ***softball***, ***baseball***, ***basketball***, ***basketball*** (yes, duplicate value).
3. Verify the content of ***ts1***.
4. Create another ***TreeSet***, called ***ts2***, to store String.
5. Use ***add()*** to add ***netball***, ***softball***, ***football***.
6. Verify the content of ***ts2***.
7. Experiment the ***TreeSets*** with ***clone***(), ***addAll***(), ***retainAll***(), ***removeAll***() and ***clear***().