(Quiz)Midterm Preparation for Week 9

Concepts

- week5 (Objects, Constructors, Encapsulation)
- week6 (Static methods/fields, Inheritance, Polymorphism, Equality and Copy Constructor)
- week7 (References/Polymorphism, Memory Management, Imports and Libraries, Serialization)
- week8 (Abstract, Interface, Anonymous Class, Lambda Expression)
- week9
  - Big-O Notation
  - Type Parameters
  - Array List
  - Linked List
  - Inner Class

EMP Session Links

1. EMP 10-22: Topics including ArrayList, Practices with Leetcode.

Concept Overview & Practice

Big-O notation

1. Big-O notation describes the limiting behavior of a function when the argument goes to a particular value or infinity (an upper bound of the function runtime).
2. Usually when we talk about the runtime of a given function, we are talking about the worst case scenario in Big-O notation.

Practice 1

1. What is the best runtime (in big-O notation) of the following codes? What is its worst runtime?

```
public int sum(int[] arr) {
  int sum = 0;
  for (int i = 0; i < arr.length; i++) {
    sum += arr[i];
  }
  return sum;
}
```

2. What is the best/worst runtime of the following codes?

```
public boolean find2D(int[][] arr, int target) {
  for (int i = 0; i < arr.length; i++) {
    for (int j = 0; j < arr[i].length; j++) {
      if (arr[i][j] == target) {
        return true;
      }
    }
  }
  return false;
}
```

Type Parameters

1. If not specified, List get function will return *Object* type item. E.g. `list.get(0)` gives you the item at index 0 in Object type.

2. Type parameter enables you to tell Java what type of items will be in the List:

```
List<E> list = new ArrayList<>();
```

where E is the type parameter that user specifies.

Practice 2

Is there an error in the following codes? If so what is it?

```
public class Pet { }
public class Dog extends Pet { }
List<Dog> pets = new ArrayList<>();
pets.add(new Pet());
pets.add(new Dog());
```

Array List

1. An array list stores items in an array, which can adjust its size as the program runs. To implement the array list yourself, you would need to implement get(int index), set(int index, Object o), and add(int index, Object o).

Practice 3

1. What is the runtime for a typical get(int index) function? What is the runtime for a set(int index, Object o) function?
2. Given the following implementation of an add function, what is its runtime?

```
public class SimpleArrayList {
  private Object[] list;
  // ...
  // ... other necessary implementation above

  public void add(Object o) {
    // obtain the correct index where the new item
    // will be added
    int indexToAdd = list.length;

    // an O(1) function to return a longer list so it
    // can fit the new item
    list = this.incrementLength();

    // assign the new item to the correct index
    list[indexToAdd] = o;
  }
}
```

3. Given the following implementation of an add function, what is its runtime?

```
public class SimpleArrayList {
  private Object[] list;
  // ...
  // ... other necessary implementation above

  public void add(int index, Object o) {
    assert index >= 0 && index <= list.length;
    Object[] newlist = new Object[list.length + 1];

    for (int i = 0; i < index; i++) {
      newlist[i] = list[i];
    }
    newlist[index] = o;
    for (int i = index + 1; i < newlist.length; i++) {
      newlist[i] = list[i - 1];
    }
    list = newlist;
  }
}
```

Linked List

1. A linked list contains a start (the first item in the list), and all the following items are connected by "links". In another word, the first item has a reference that points to the second item, then the second item has a reference to the third item, and so on:

```
O1(start) -> O2 -> O3 -> null
```

2. If we add item to the front of a linked list, the runtime will be O(1); if we add item to the back of a linked list, normally it is O(n); if we want to set/get an item at a certain index, normally the runtime is O(n).

Practice 4

Come up with an O(1) function to add item to the end of a linked list.

▶ Spoiler!

```
public class SimpleLinkedList {
  private class Item {
    private Object value;
    private Item next;
    // ... necessary implementations
  }

  private Item start;
  private Item end;
  private int size;

  // ... necessary implementations
  public SimpleLinkedList(Object[] values) {
    assert values != null;
    for (int i = 0; i < values.length; i++) {
      addToEnd(values[i]);
    }
  }

  // function to add item to the end of the linked list
  public void addToEnd(Object o) {
    if (start == null) {
      start = new Item(o, null);
      end = start;
    } else {
      Item prevEnd = end;
      Item newEnd = new Item(o, null);
      prevEnd.next = newEnd;
      end = newEnd;
    }
  }
}
```

Inner Class

1. We could declare an inner(nested) class to be private so the inner class is "hidden" from the outer world, thus increasing encapsulation (preventing unauthorized parties to directly access certain elements).
2. You can have nested class, interface.
3. You can access inner class's variables in the outer class, even if they are declared as private.
4. To be able to declare a nested class instance outside of the outer class, we need the static keyword. This is called static nested class.

```
public class Outer {
  public static class Inner { }
}
Outer.Inner inner = new Outer.Inner();
```