Exam 1 Summer 2021 Solutions

Q1. True/False and Multiple-Choice

Q1.1. `LinkedLists` have better cache performance than `ArrayLists`. True/False

Q1.2. For the frequent insert and delete operations, `LinkedList` performs better than `ArrayList`. True/False

Q1.3. A class extending an abstract class must be made abstract if abstract method(s) from the super class are not defined in the subclass. True/False

Q1.4. The class `Shape` is an abstract class with a default constructor. Which of the following are valid?

```
Shape s = null;
Shape t[10];
Shape s2 = new Shape();
```

Q1.5. The `Student` class has `id` of type `int` and `name` of type `String`, two instance variables. Fill in the blanks to complete the `equals` method of the `Student` class. Two students objects are equal if they have the same `id` and `name`.

```
public class Student{
    private int id;
    private String name;
    @Override
    public boolean equals(Object arg){

        if (!(arg instanceof Student)) return false;

        if(this == arg) return true;

        Student s = (Student) arg;

        if(id == s.id && name.equals(s.name)){
            return true;
        }
        else{
            return false;
        }
    }
}
```

Q1.6. `protected` and `default` access specifiers have same visibility. True/False

Q1.7. A method defined in a superclass is re-defined in a subclass with an identical method signature is called method overloading.

Q1.8. What is the output of the code below?

```
class Base{
    final public String show() {
        return “Base”;
    }
}
class Child extends Base{
    public String show() {
        return “Child”;
    }
}
class Main{
    public static void main(String[] args) {
        Base b = new Child();
        b.show();
    }
}
```

Base
Child
Compiler Error
Runtime Error

Q1.9. Abstract classes can have:

Abstract methods
Constructors
Private instance variables
Method implementation (method body)

Q1.10. The `ArrayList` must be sorted if we want to use the binary search. True/False

Q2. Short Answer

`ArrayBag` implementation: https://github.com/anwarmamat/cmsc132/blob/master/bag/Bag.java

Q2.1. Explain why we need the `resize` method in the `ArrayBag` implementation, but do not need it in the `LinkedBag` implementation.

ArrayBag relies on an array to store data. Since array size is fixed, we need to resize the array to increase capacity. However, LinkedBag stores data using a linked list. Since linked lists grow dynamically, we do not need a resize method.

Q2.2. In the resize method of the `ArrayBag`, we doubled (`capacity = capacity * 2`) the size of the array, instead of incrementing by 1 (`capacity = capacity + 1`). Explain the advantages and disadvantages of doubling the array size.

Doubling the capacity is more efficient because we do not need to resize as often, but this leads to wasted space.

Q2.3. What are the advantages of `immutable` objects in Java?

Immutable objects cannot be changed. Prevents unexpected side-effects, better encapsulation, thread safety, can be used in HashMaps, etc.

Q3.

We implemented a `Bag` container and inserted integers into the `bag` instance.

```
Bag<Integer> bag = new Bag<>();
for(int i = 1; i <= 5; i++){
   bag.insert(i);
}
```

Above code worked and we can see the size of `bag` is 5. Now, we want to print the integers in the `Bag` with the following code:

```
for(Integer k:  bag){
     System.out.println(k);
}
```

However, java compiler shows an error. What do you think the reason for the compiler error? How should we fix the error?

The bag needs to be iterable. We can fix this issue by implementing the Iterable interface.

Q4.

The method `length` calculates the length of a `LinkedList`. `head` points to the first node. `Node` is defined as:

```
class Node {
    int val;
    Node next;
}
```

Fill in the blanks to complete the function:

```
public int length(Node head){
  Node cur = head;
  int len = 0;
  while(cur != null){
         len++;
         cur = cur.next;
  }
  return len;
}
```

Q5. What's the input?

```
public void foo() {
  Node h = first; // first node of the LinkedList
  while(h != null) {
    if((Integer)h.data % 2 == 0 ){
      System.out.print(h.data+",");
      h = h.next;
    }
    h = h.next;
 }
```

List the elements of the `LinkedList` so that calling `foo` prints `2,4,6`.

1, 2, 3, 4, 5, 6, 7

Q6. What's the input?

```
public int foo() {
 int s = 0;
 Node cur = first;
 for(int i = 0; i < 5; i++){
    s = s + cur.data;
    cur = cur.next;
 }
 return s;
}
```

When we called `foo` on a linked list, where `first` points to the first node of the list, it returned 10. What are the elements of the linked list.

Sum of first 5 elements must be 10. For example: 0, 1, 2, 3, 4

Q7. Copy Constructor

We want to implement a constructor in the `ArrayBag` we implemented in class. This constructor receives another `ArrayBag` as an argument and creates a new `ArrayBag` by copying the argument's content. For simplicity, you can assume the `ArrayBag` contains only `Integers` or `Strings`. Here is the implementation of the `ArrayBag` (https://github.com/anwarmamat/cmsc132/blob/master/bag/Bag.java)

```
public class ArrayBag<E>{
  private E[] items;
  private int N;
  private in capacity = 10;
}

public ArrayBag(ArrayBag bag){
  N = bag.N;
  capacity = bag.capacity;
  items = new E[capacity];
  for (int i = 0; i < items.length; i++) {
    items[i] = bag.items[i];
  }
}
```

Q8. Doubly Linked List

```
private class Node{
     private E data;
     private Node next;
     private Node prev;
}
```

In a doubly linked list, p references a random node, which is not the head neither the tail. Write code to delete the node. You have access to p, but you are not allowed to start from head.

```
p.prev.next = p.next;
p.next.prev = p.prev;
```

Q9.

Explain what the function `bar` will do if the node `first` points to the first node of a linked list.

```
public Node bar() {
  Node h1 = first;
  Node h2 = first;
  while(h1 != null) {
     h1 = h1.next;
     if(h1 == null) {
      return h2;
     }else {
      h1 = h1.next;
     }
     h2 = h2.next;
  }
  return h2;
}
```

Returns the midpoint/middle node of the linked list.

Q10. ArrayBag Shrink

In the `ArrayBag` implementation, we increased the size array by calling `resize` when the `ArrayBag` is full. Now, you want to implement `shrink`, the opposite of `resize`. When `shrink` is called, we first check if the `ArrayBag` is using less than 20% of its capacity. If it does, we create an array with half the capacity of the current array and move the `ArrayBag` contents to the new array.

```
public class ArrayBag<E>{
   private int N;
   private int capacity;
   private E[] items;
}
```

Complete the `shrink` method below:

```
private void shrink(){
   if(N < capacity * 0.2){
      capacity = capacity / 2;
      E[] temp = new E[capacity];
      for (int i = 0; i < capacity; i++) {
         temp[i] = items[i];
      }
      items = temp;
   }
}
```

Q11. Single Linked List Insert

The following method `insert(E item)` insert a node at the tail of a `LinkedBag` we implemented in class. Fill in the blanks to complete the code.

```
public void insert(E item){
    if(first == null){
        first = new Node(item);
        return;
    }
    Node cur = first;
    while(cur.next != null){
         cur = cur.next;
    }
    cur.next = new Node(item);
}
```