University of Bahrain

College of Information Technology

Department of Computer Science

Second Semester, 2017-2018

**ITCS214 / ITCS215 / ITCS216 (Data Structures)**

#### Test I

Date: 22/03/2018 Time: 17:00 - 18:15

**STUDENT NAME** (Uppercase characters)

|  |
| --- |
|  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **2** | 0 |  |  |  |  |  |  |

**STUDENT ID #**

|  |
| --- |
|  |

**SECTION #**

NOTE: THERE ARE EIGHT **(8) PAGES** IN THIS TEST

ONLY ONE SOLUTION WILL BE CONSIDERED FOR EACH QUESTION

|  |  |  |  |
| --- | --- | --- | --- |
| QUESTION # | MARKS | | COMMENTS |
| 1 | 10 |  |  |
| 2 | 12 |  |  |
| 3 | 8 |  |  |
| TOTAL | 30 |  |  |

**Question 1 [10 Marks] [Inheritance]**

Consider the following abstract class definition:

abstract class Item

{

private String itemName;

private int itemNum;

private double price;

public Item( ){

this(“”, 0, 0.0);

}

public Item(String name, int num, double pr){

itemName = name;

itemNum = num;

price = pr;

}

public void setItemName(String name){

itemName = name;

}

public void setItemNum(int num){

itemNum = num;

}

public void setPrice(double pr){

price = pr;

}

public string getItemName( ){

return itemName;

}

public int getItemNum( ){

return itemNum;

}

public double getPrice( ){

return price;

}

public String toString( ){

return(“Item Name: ” + itemName + “Item Num: ” + itemNum +

“Price: ” + price);

}

public abstract double totalPrice();

} // end Item

Write a concrete class (with implementation of all methods) called **StoreItem,** which inherits the properties of class **Item.** This new class has the following additional members:

Data fields (private): make(string), quantity (int)

Methods (public):

* Default constructor – without any parameters
* constructor with 5 parameters.
* set and get methods for both data fields separately,
* totalPrice: returns price multiplied by quantity.
* toString: returns String equivalent of all attributes (including that of **Item**).

**Answer Q1**

public class StoreItem extends Item

{

private String make;

private int quantity;

//Default constructor

public StoreItem( ){

this(“”, 0, 0.0, “”, 0);

}

//Constructor with parameters

public StoreItem(String name, int num, double pr, String mak,

int qty)

{

super(name, num, pr);

make = mak;

quantity = qty;

}

public void setMake(String mak){

make = mak;

}

public void setQuantity(int qty){

quantity = qty;

}

public String getMake( ){

return make;

}

public int getQuantity( ){

return quantity;

}

public double totalPrice(){

return (getPrice()\* quantity);

}

public String toString( ){

return (super.toString( ) + “Make: ” + make + “Quantity: ” + quantity);

}

}

**Question 2 [7 + 5 Marks] [Array List]**

Consider the generic class **KWArrayList(**as discussed in the lectures) having following data fields (private):

private static final int INITIAL\_CAPACITY = 10; // The default initial capacity

private E[] theData; // The underlying data array

private int size ; // The current size

private int capacity ; // The current capacity

This class has the following methods:

|  |  |
| --- | --- |
| **Method** | **Behavior** |
| public KWArrayList ( ) | Default Constructor with capacity = INITIAL\_CAPACITY |
| public KWArrayList (int cap ) | Constructor with capacity = cap |
| **public int size()** | Returns current size |
| **boolean contains(E obj)** | Checks whether the given object obj is present in the list. If it is there then it returns true else it returns false. |
| **clear()** | Removes all the elements of the array list and make it empty |
| isEmpty() | Checks whether list is empty or not |
| public boolean add ( E anEntry) | Adds object anEntry at the end of the list and returns true. |
| public void add (int index, E anEntry) | Adds object anEntry in the list at the location given by index |
| public E get (int index) | Returns the element of the list at position given by index |
| public E set (int index, E newValue) | Updates the element at position index by newValue and returns the old value |
| public E remove (int index) | Removes the element at position index and returns the element being removed |
| public boolean remove (E obj) | Removes the first occurrence of the object obj from the list, if present and returns true, else returns false. |
| private void reallocate () | Private method to expand the array by allocating a new array of double the previous capacity. Called if the list becomes full |
| public int indexOf(E obj) | Returns the index of the first occurrence of the specified element obj in this list, or -1 if this list does not contain the element. |
| public String toString() | Returns the String equivalent of the list object |

1. Write a method to be included in class **KWArrayList** to insert a new item **anEntry** at location size/2 of the list, if it is not already in the list and return true. If the item **anEntry** already exists in the list, then do not insert it and return false. If the list is empty, insert **anEntry** at location 0 and return true.

Method heading: **public boolean insertMid(E anEntry)**

Do the necessary shifting of elements in the array and reallocate bigger array, if array

becomes full. You can call **reallocate** method of class **KWArrayList** in your method,but do not call any other method of the class **KWArrayList** in your method.

**Answer Q2 (A)**

**public boolean insertMid(E anEntry)**

{

if(size == 0){

theData[0] = anEntry;

size++;

return true;

}

*//search anEntry in the array*

boolean found = false;

for(int i = 0; i < size; i++) {

if ( anEntry.equals(theData[i]) )

found = true;

if (found)//item already exists in the list

return false;

if (size == capacity) *// Make sure there is space to add new item*

reallocate();

*// shift data in the array to make space for the new item*

for (int i = size; i > size/2; i--) {

theData[i] = theData[i-1];

}

theData[size/2] = anEntry; *// insert item*

size++;

return true;

}

1. Write an application class called **ArrayListExample** having only **main** method that has two objects **list1**and**list2** of type **KWArrayList**. If both the lists **list1**and**list2** have exactly the same elements, but in any order, then output *“Elements in both lists are same”*, else output *“Elements in both lists are not same”*.

Note: Complete this application by calling methods of the class **KWArrayList** given at the

beginning of the question.

Example 1: list1: 10 5 7 8 2 list2: 7 8 5 10 2

In this case, the method should output*“Elements in both lists are same”* as all the

elements of both lists are same. But, the order is different.

Example 2: list1: 10 5 **7** 8 **3** list2: **12** 8 5 **11** 10

In this case, the method should output *“Elements in both lists are not same”*, as some

elements of list1 and list2 are not same.

public class ArrayListExample

{

public static void main(String [ ] args)

{

KWArrayList<Integer> list1 = new KWArrayList<Integer>( );

KWArrayList<Integer> list2 = new KWArrayList<Integer>( );

Scanner in = new Scanner(System.in);

System.out.println(“Enter 5 integers to be added in list1: ”);

for(int i = 0; i < 5; i++)

list1.add( in.nextInt( ) );

System.out.println(“Enter 5 integers to be added in list2: ”);

for(int i = 0; i < 5; i++)

list2.add( in.nextInt( ) );

boolean equals = true;

// complete the remaining application

**// Answer Q2 (B)**

for(int i = 0; i < list1.size(); i++)

//**OR** for(int i = 0; i < 5; i++)

{

int item = (Integer)list1.get(i);

if ( list2.contains(item)) == false )

//**OR** if(list2.indexOf(item) == -1)

{

equals = false;

break;

}

} // end for

if(equals)

System.out.println(“Elements in both lists are same”);

else

System.out.println(“Elements in both lists are not same”);

} // end main

}

**Question 3 [8 Marks] [Single Linked List]**

Write a method called **copyAlternate** to be included in the class **SingleLinkedList** that accepts a parameter **list1** of type **SingleLinkedList**. If "this" list is empty, it will return false, otherwise, it will copy alternative nodes from "this" list to **list1** and returns true. Assume that **list1** is initially empty. Do not call any method of class **SingleLinkedList**.

Method heading:

**public boolean copyAlternate(SingleLinkedList<E> list1)**

Note: Copy here means creating and adding new nodes in **list1**. You can use constructor of class **Node** to create new nodes.

Example :

**Before method call:**  "this" list: 7 8 10 9 6

list1: ( empty )

**After method call:** "this" list: 7 8 10 9 6

list1: 7 10 6

**Answer Q3**

public boolean copyAlternate (SingleLinkedList<E> list1)

{

Node<E> current, newNode, last;

if (head == null) // If “this list” is empty

return false;

current = head; // Reference to the original list ‘”this list”

list1.head = new Node<E>(current.data); //First node of copy list

last = list1.head; // Reference to the last node of list1,

//used to connect new nodes

(list1.size)++;

current = current.next; // go to the next node

if(current != null) //skip this node

current = current.next;

// Copy remaining nodes

while (current != null)

{

newNode = new Node<E>(current.data);

last.next = newNode; // Link last node to the new node

last = newNode;

(list1.size)++;

current = current.next;

if(current != null) //skip this node

current = current.next;

} // end while

return true;

}

[Empty Page]