**7 Packages, Try/Catch, Exceptions. Junit testing. Lab 6 assignment**

**8 (Midterm Exam)**

**9 JDBC Database Access. DAO Design Pattern. Refactoring and DRY (Don’t Repeat Yourself!). Lab 7 assignment**

**Java Packages**

**Creating and Using Packages**

[**http://docs.oracle.com/javase/tutorial/java/package/packages.html**](http://docs.oracle.com/javase/tutorial/java/package/packages.html)

**[Exercise ]**

Assume that you have written some classes. Belatedly, you decide that they should be split into three packages, as listed in the table below. Furthermore, assume that the classes are currently in the default package (they have no package statements).

|  |  |
| --- | --- |
| **Package Name** | **Class Name** |
| mygame.server | Server |
| mygame.shared | Utilities |
| mygame.client | Client |

a. What line of code will you need to add to each source file to put each class in the right package?  
Answer 1a: The first line of each file must specify the package:

In Client.java add:

package mygame.client;

In Server.java add:

package mygame.server;:

In Utilities.java add:

package mygame.shared;

[**http://www.javatpoint.com/package**](http://www.javatpoint.com/package)

**Java Try/Catch, Exceptions**

**Java Exceptions**

public void openFile(){

try {

// constructor may throw FileNotFoundException

FileReader reader = new FileReader("someFile");

int i=0;

while(i != -1){

//reader.read() may throw IOException

i = reader.read();

System.out.println((char) i );

}

reader.close();

System.out.println("--- File End ---");

} catch (FileNotFoundException e) {

//do something clever with the exception

} catch (IOException e) {

//do something clever with the exception

}

}

**Data Structures - The Linear Search Algorithm**

**(1)** ( **Linear Search** )

Techniques used to search arrays include the linear search, also known as a sequential search. For the linear search approach, a search is performed through an element by element comparison basis, from either the beginning to the end or from the end to the beginning. For this type of model the list may be in any order, i.e. the list does not have to be in a sorted order.

Consider the code below, which involves a linear search, and the corresponding output.

|  |
| --- |
| #include <iostream.h>  void main()  {  const int SIZE = 6;  int array[SIZE] = { 1, 2, 3, 4, 5, 6 };  int key;  char found = 'n';  cout << "Please enter the number you wish to search. ";  cin >> key;    for ( int index = 0; index < = SIZE - 1 && found = = 'n'; index + + )  {  if (array[index] = = key)  {  found = 'y';  cout << "the number is located at index " << index << endl;    }  }    if(found == 'n')  {  cout << "item not found" << endl;  }  } |

**Program Output**:

Please enter the number you wish to search. 5

the number is located at index 4

|  |
| --- |
|  |

Based upon the above program, answer each of the following.

\_\_\_\_\_ The given list ( array ) in the above program does not have to be in sorted order.

(a) True (b) False

\_\_\_\_\_ In the above code, the found variable is originally set to 'y' for Yes.

(a) True (b) False

\_\_\_\_\_ In a linear search, the desired item to search is often referred to as a(n) \_\_\_\_\_ .

(a) flag (b) index (c) key (d) array (e) sort

**(2)** Arrange each of these lines of the linear search pesudocode process in their proper order by placing, in the spaces provided, a **(1)** next to the first step, a **(2)** next to the second step, and so on.

\_\_\_4\_\_ Return –1 because the item is not found

\_\_\_1\_\_ Compare the list item with a desired item ( which will be the key )

\_\_\_3\_\_ Else continue searching for the item, unless the end of the array is reached

\_\_\_2\_\_ If the item was found, return the index value of the current item

**(3)** The following program uses a function that returns the location of a key in a given list. The value − 1 , for false, is returned if the value is not found.

|  |
| --- |
| #include <iostream.h>  linearSearch( int list[ ], int size, int key );  void main()  {  int key = 12;  int list[8] = {5, 4, 8, 10, 12, 18, 0, 13};    cout << "(a -1 indicates the item was not found)" << endl;    cout << "the number is located in position "  << linearSearch(list, SIZE, key) << endl;  }  int linearSearch(int list[], int size, int key)  {  for(int i = 0; i < size; i++)  {  if (list[i] = = key)  return i + 1;  }  return -1;  } |

Referring to the above code, identity each of the following.

(a) \_\_\_\_12\_\_\_\_\_\_ the search key

(b) \_\_\_\_8\_\_\_\_\_\_ the array ( list ) size

(c) \_\_linear search\_\_\_\_\_\_\_\_ the applied search technique

(d) \_\_\_\_5\_\_\_\_\_\_ the number of necessary comparisons to locate the key

The following program uses a function named Search() which will search for a name to match an account number.

|  |
| --- |
| **class** LinearSearch  {  **static** **int**[] *account* = {15, 20, 25, 35, 50, 60, 75, 90, 95};  **static** String[] *client* = **new**  String[]{"Allen", "Bensen", "Davies", "Everly", "Jensen", "Mann", "Nolan", "Oh", "Peters"};  **public** **static** **void** main(String args[])  {  *Search*();  }  **public** **static** **void** Search()  {  **char** found = 'n';  **int** number = 0;  System.*out*.println( "Enter an Account Number ---> ");  //input number;  **for**(**int** i = 0; i < 9 && found == 'n'; i++)  {  **if**(number==*account*[i])  {  found = 'y';  System.*out*.println( "\n Client: " + *client*[i] + "\n" );  }  }  **if**(found == 'n')  {  System.*out*.println("\nClient not found - check number\n" );  }  }  } |

Modify the above program to write a search routine to find the room number when a given seminar topic is entered. Assume that the following arrays have been declared and initialized.

int room[5] = { 100, 200, 300, 305, 400 };

String topic[5][8] = { “UNIX”, “FORTRAN”, “C++”, “VB”, ”C” };

**Data Structures - The Binary Search Algorithm**

For a Linear Search, the average number of required comparisons is *N* / 2 where *N* is the size of the list. What would be the average number of comparisons using a Linear Search with a list of 100 items? What happens if *N* = 200 ?

What would be the average number of comparisons using a Linear Search with a list of 50 items? What happens if *N* = 250 ?

The Binary Search method is more efficient that the Linear Search method since each iteration reduces the search by a factor of two, i.e. for an array of size *N* ,the Binary Search

has log 2 *N* number of required comparisons whereas the Linear Search has *N* required comparisons. In Big *O* notation, this would be expressed as *O* ( log 2 *N* ) for the Binary Search and *O* ( *N* ) for the Linear Search.

For example, if an array has *N* = 1,000 elements, the Linear Search would require 1,000 iterations, whereas the Binary Search would not require more than 10 since:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| log 2 *N* = log 2 1000 = | + log 10 1000 | = | +  3 | ≈ 10 |
| +  log 10 2 | +  0.301030 |

What is the maximum number of comparisons needed for a list consisting of 2,000 elements if you use a Binary Search routine?

An array need not be in sort order to perform a Binary Search on the array.

(a) True (b) False

( **Binary Search Algorithm** )

The mechanism of the Binary Search algorithm basically is this: on each iteration of the main loop, the middle element a [ middle ] of the sub - array from a [ left ] to

a [ right ] is checked for the key. If it is not found, then either the left hand side is discarded by resetting left = location + 1 , or the right half is discarded by resetting right = location – 1 , according to whether a [ location ] < key.

Consider the following array declaration and initialization:

int a[size] = { 5, 10, 22, 32, 45, 67, 73, 98, 99, 101 }

How many comparisons would it take to find the value 73 ? Assume that you are performing a binary search.

Fill in the tables below and show the values for left, middle, and a [ middle ], each step of the way. The initial values are provided for you.

|  |  |  |  |
| --- | --- | --- | --- |
| **left** | **middle** | **right** | **a [ middle ]** |
| **00** | **04** | **09** | **45** |
| **05** | **07** | **09** | 98 |
| **05** | **06** | **07** | **073** |
|  |  |  |  |
|  |  |  |  |

What index number is the item 73 at? \_\_\_\_\_6\_\_\_\_\_

**Java JDBC**

[**http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html**](http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html)

**Simple JDBC Example**

**To connect a java application to a database we perform the following steps:**

**1. Load the driver of particular database you are using.**

**2. Create a Connection object to get a connection.**

**3. Create a statement object to for executing query.**

**4. Create a Resultset object to store the query.**

**5. Finally close the Connection, Statement and Resultset objects.**

**At first create table named student in MySQL database inset values into it as.**

**Creating A table**

**CREATE TABLE `student` (**

**`rollno` int(11) NOT NULL,**

**`name` varchar(50) default NULL,**

**`course` varchar(20) default NULL,**

**PRIMARY KEY (`rollno`)**

**)**

**Inserting Values**

**INSERT INTO STUDENT(rollno,name,course) VALUES(1,'Tommy',T.Tech') ;**

**A Simple example is given below to connect a java application to a SQL database**

[**http://www.javatpoint.com/java-jdbc**](http://www.javatpoint.com/java-jdbc)

[**http://stackoverflow.com/questions/4320623/connecting-a-microsoft-access-database-to-java-using-jdbc-and-compiling**](http://stackoverflow.com/questions/4320623/connecting-a-microsoft-access-database-to-java-using-jdbc-and-compiling)

[**http://www.codeproject.com/Articles/35018/Access-MS-Access-Databases-from-Java**](http://www.codeproject.com/Articles/35018/Access-MS-Access-Databases-from-Java)

**Java with Junit**

[**http://junit.org/**](http://junit.org/)

[**http://www.tutorialspoint.com/junit/**](http://www.tutorialspoint.com/junit/)