**Exp No: 01 DATE: 28/09/2021**

**DECISION MAKING, BRANCHING AND LOOPS IN JAVA**

**AIM:**

To implement a java program to exhibit branching and looping mechanisms in java.

**THEORY:**

**Problem Statement:**

Write a program that creates a class Result. the program allows the user to enter either the marks in each of the 6 subjects or the aggregate score of all subjects and the program calculates the percentage obtained by the student based on the following criteria:

1. if the percentage is less than 40 print the message repeat the semester

2. if the percentage is between 40 to 59 than pass class

3. 60 to 74 then first class

4. if it is above 75 then distinction

The program should also tell the end user if he or she has failed in any of the individual subjects and suggest remedial classes.

A programming language uses control statements to control the flow of execution of program based on certain conditions. These are used to cause the flow of execution to advance and branch based on changes to the state of a program.

The conditional statements in [JavaScript](https://www.javatpoint.com/javascript-tutorial) are listed below:

* if statement
* if….else statement
* if….else if….statement
* nested if statement
* switch statement

**The if statements**

It is one of the simplest decision-making statement which is used to decide whether a block of JavaScript code will execute if a certain condition is true.

**Syntax**

**if (condition)**

**{ // block of code will execute if the condition is true }**

If the condition evaluates to true, the code within **if statement** will execute, but if the condition evaluates to false, then the code after the end of **if statement (after the closing of curly braces)** will execute.

### The if….else statement

An **if….else statement** includes two blocks that are **if block** and **else block.** It is the next form of the control statement, which allows the execution of JavaScript in a more controlled way. It is used when you require to check two different conditions and execute a different set of codes. The **else statement** is used for specifying the execution of a block of code if the condition is false.

**Syntax**

if (condition)

{ // block of code will execute if the condition is true }

else

{ // block of code will execute if the condition is false }

If the condition is true, then the statements inside **if block** will be executed, but if the condition is false, then the statements of the **else block** will be executed.

**The if….else ladder statement**

It is used to test multiple conditions. The if statement can have multiple or zero else if statements and they must be used before using the else statement. You should always be kept in mind that the else statement must come after the else if statements.

Syntax

if (condition1)

{ // block of code will execute if condition1 is true }

else if (condition2)

{ // block of code will execute if the condition1 is false and condition2 is true }

else

{ // block of code will execute if the condition1 is false and condition2 is false }

### The nested if statement

It is an if statement inside an if statement.

**Syntax**

if (condition1)

{

Statement 1; //It will execute when condition1 is true

if (condition2)

{ Statement 2; //It will execute when condition2 is true }

else

{ Statement 3; //It will execute when condition2 is false }

}

### The switch statement

It is a multi-way branch statement that is also used for decision-making purposes. In some cases, the **switch statement** is more convenient than **if-else statements.** It is mainly used when all branches depend upon the value of a single variable. It executes a block of code depending upon the different cases.

**default:** It specifies some code to run when there is no case match. There can be only a single default keyword in a switch. It is also optional, but it is recommended to use it as it takes care of unexpected cases.

**Syntax**

switch(expression)

{

case value1: //code to be executed;

break; //optional

case value2: //code to be executed;

break; //optional

......

default:

code to be executed if all cases are not matched;

}

**Jump Statements**

Java supports 3 jump statements: break, continue and return. These 3 statements transfer control on to other parts of the program.

1.Break: in Java, break is majorly used for terminating a sequence in a switch statement and to exit a loop.

2.Continue: Sometimes it is useful to force an early iteration of a loop. You might want to continue running the loop but stop processing the remainder of the code in its body for a particular iteration.

3.Return: the return statement is used to explicitly return from a method. That is, cause a program control to transfer back to the caller of the method.

**PROGRAM:**

package result;

import java.util.Scanner;

public class Result

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int choice;

float java,wt,dbms,os,clIpr,osha,aggregate,percentage = 0.0f;

do

{

System.out.println("\n1.Enter marks of 6 subjects\n2.Enter aggregate score\n3.Exit");

System.out.print("\nEnter your choice: ");

choice=sc.nextInt();

switch(choice)

{

case 1:

System.out.print("\nEnter marks of JAVA: ");

java=sc.nextFloat();

System.out.print("Enter marks of Web Tech: ");

wt=sc.nextFloat();

System.out.print("Enter marks of DBMS: ");

dbms=sc.nextFloat();

System.out.print("Enter marks of OS: ");

os=sc.nextFloat();

System.out.print("Enter marks of CL & IPR: ");

clIpr=sc.nextFloat();

System.out.print("Enter marks of OSHA: ");

osha=sc.nextFloat();

percentage=(java+wt+dbms+os+clIpr+osha)/6;

System.out.println("\nYour percentage is: "+percentage+"%");

if(java<40)

System.out.println("\nYou have failed in JAVA, please join remedial classes");

if(wt<40)

System.out.println("\nYou have failed in Web Tech, please join remedial classes");

if(dbms<40)

System.out.println("\nYou have failed in DBMS, please join remedial classes");

if(os<40)

System.out.println("\nYou have failed in OS, please join remedial classes");

if(clIpr<40)

System.out.println("\nYou have failed in CL & IPR, please join remedial classes");

if(osha<40)

System.out.println("\nYou have failed in OSHA, please join remedial classes");

break;

case 2:

System.out.print("\nEnter aggregate score: ");

aggregate=sc.nextFloat();

percentage=aggregate/6;

System.out.println("\nYour percentage is: "+percentage+"%");

break;

case 3:

return;

default:

System.out.println("Invalid input");

break;

}

}while(choice!=3);

if(percentage>=75)

System.out.println("\nClass: Distinction");

else if(percentage>=60 && percentage<=74)

System.out.println("\nClass: First class");

else if(percentage<40 || percentage>59)

System.out.println("\nYou have failed repeat the semester");

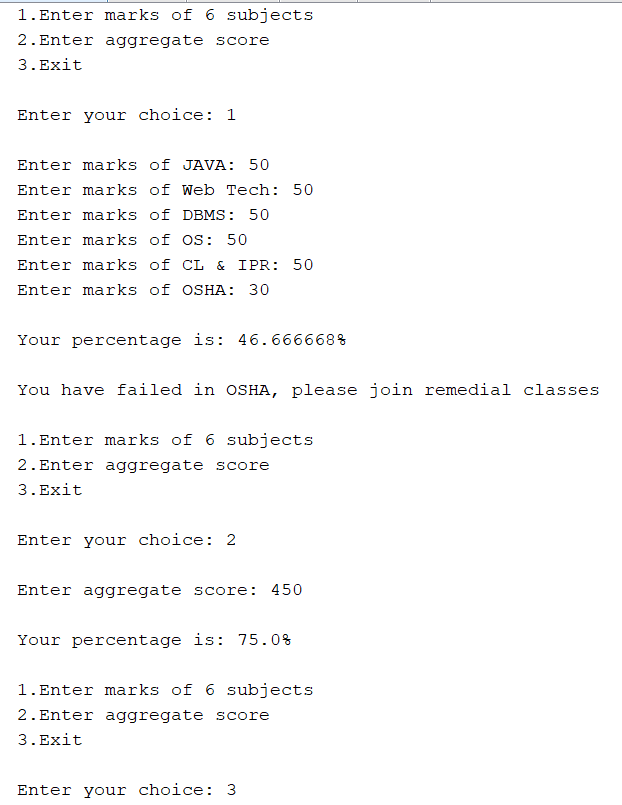
else

System.out.println("\nClass: Pass class");

}

}

**OUTPUT:**

****

**CONCLUSION:**

The program to exhibit branching and looping mechanism in java was implemented successfully.

**Exp No: 02 DATE: 12/10/2021**

**MATRICES IN JAVA**

**AIM:**

To implement a java program for performing various matrix operations.

**THEORY:**

**Problem Statement:**

Write a program to create a class called matrix. Perform the following operations:

1. Addition of 2matrices.
2. Multiplication of scalar to a matrix.
3. Transpose of a matrix.
4. Multiplication of 2 matrices.

**Arrays in JAVA:**

Java array is an object which contains elements of a similar data type. Additionally, the elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java.

Advantages

●Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.

●Random access: We can get any data located at an index position.

Disadvantages

●Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Arrays in java

●Single Dimensional Array

●Multidimensional Array

Creating an array

Creation of an array involves the following 3 steps:

1. Declaring the array.

2. Creating memory locations

3. Printing values into the memory location.

Declaration of arrays

Arrays in Java may be declared in 2 forms:

Form 1: type arrayname[];

Form 2: type []arrayname;

Creation of arrays

Java allows us to create arrays using new operator only, as shown below

Arrayname = new type[size];

Initialization of arrays

The final step is to put into the array created. This process is known as initialization.

Arrayname[subscript] = value;

Different operations on Matrices:

* Matrix Addition:

The addition of 2 matrices Am\*n and Bm\*n gives a matrix Cm\*n

The order of matrix A,B and A+B is always same.

* Matrix Multiplication:

The multiplication of 2 matrices Am\*n and Bp\*q gives a matrix Cm\*p

i.e. column of A should be equal to row of B.

* Transpose:

Transpose of a matrix is produced by swapping the rows with column.

**PROGRAM:**

package matrix;

import java.util.\*;

class Arrays

{

Scanner sc = new Scanner(System.in);

void add()

{

System.out.print("Enter the number of rows of matrix 1 : ");

int a = sc.nextInt();

System.out.print("Enter the number of columns of matrix 1 : ");

int b = sc.nextInt();

System.out.print("Enter the number of rows of matrix 2 : ");

int c = sc.nextInt();

System.out.print("Enter the number of columns of matrix 2 : ");

int d = sc.nextInt();

int m1[][] = new int[a][b], m2[][] = new int[c][d], add[][] = new int[a][b];

System.out.print("Enter matrix 1 : ");

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

for(int j=0; j<b; j++)

m1[i][j]=sc.nextInt();

System.out.print("Enter matrix 2 : ");

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

for(int j=0; j<d; j++)

m2[i][j]=sc.nextInt();

if(a!=c && b!=d)

System.out.println("Addition not possible!");

else

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

for(int j =0; j<b; j++)

add[i][j] = m1[i][j] + m2[i][j];

System.out.println("The sum of matrices is: ");

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

{

for(int j =0; j<b; j++)

System.out.print(add[i][j]+"\t");

System.out.println();

}

}

void mulsc()

{

System.out.print("Enter the number of rows : ");

int a = sc.nextInt();

System.out.print("Enter the number of columns : ");

int b = sc.nextInt();

System.out.print("Enter the Scalar Number : ");

int s = sc.nextInt();

int sm[][]= new int[a][b];

System.out.print("Enter matrix : ");

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

for(int j=0; j<b; j++)

sm[i][j]=sc.nextInt();

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

for(int j=0; j<b; j++)

sm[i][j] = sm[i][j] \* s;

System.out.println("The matrix on scalar multiplication is : ");

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

{

for(int j =0; j<b; j++)

System.out.print(sm[i][j]+"\t");

System.out.println();

}

}

void transpose()

{

System.out.print("Enter the number of rows : ");

int a = sc.nextInt();

System.out.print("Enter the number of columns : ");

int b = sc.nextInt();

int ab[][]= new int[a][b];

int tr[][]= new int[b][a];

System.out.print("Enter matrix : ");

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

for(int j=0; j<b; j++)

ab[i][j]=sc.nextInt();

System.out.println("The transpose of a matrix is : ");

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

{

for(int j =0; j<a; j++)

System.out.print(ab[j][i]+"\t");

System.out.println();

}

}

void mul()

{

System.out.print("Enter the number of rows of matrix 1 : ");

int a = sc.nextInt();

System.out.print("Enter the number of columns of matrix 1 : ");

int b = sc.nextInt();

System.out.print("Enter the number of rows of matrix 2 : ");

int c = sc.nextInt();

System.out.print("Enter the number of columns of matrix 2 : ");

int d = sc.nextInt();

int m1[][] = new int[10][10], m2[][] = new int[10][10], mul[][] = new int[10][10];;

System.out.print("Enter matrix 1 : ");

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

for(int j=0; j<b; j++)

m1[i][j]=sc.nextInt();

System.out.print("Enter matrix 2 : ");

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

for(int j=0; j<d; j++)

m2[i][j]=sc.nextInt();

if(a!=d)

System.out.print("Multiplication is not possible!");

else

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

for(int j =0; j<d; j++)

for(int k=0; k<c; k++)

mul[i][j] += m1[i][k] \* m2[k][j];

System.out.println("The matrix on multiplication is: ");

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

{

for(int j =0; j<d; j++)

System.out.print(mul[i][j]+"\t");

System.out.println();

}

}

}

public class Matrix

{

public static void main(String[] args)

{

Scanner s = new Scanner(System.in);

int n;

do

{

Arrays m = new Arrays();

System.out.print("\n1. Addition of 2 matrices\n2. Multiplication of Scalar to a matrix\n3. Transpose of a matrix\n4. Multiplication of 2 matrices\n5. Exit\nEnter your choice : ");

n = s.nextInt();

switch(n)

{

case 1:

m.add();

break;

case 2:

m.mulsc();

break;

case 3:

m.transpose();

break;

case 4:

m.mul();

break;

case 5:

break;

default:

System.out.println("wrong choice!");

break;

}

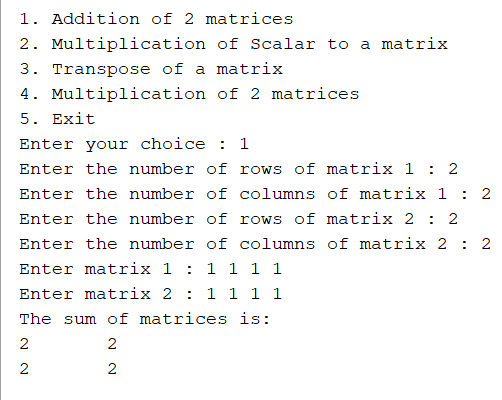
}while(n!=5);

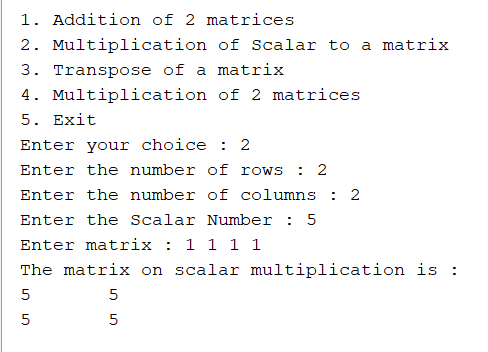
}

}

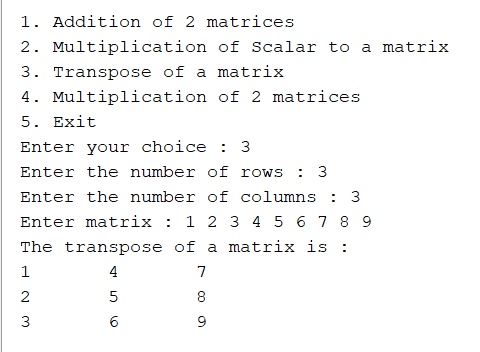
**OUTPUT:**

**Addition of 2 matrices**

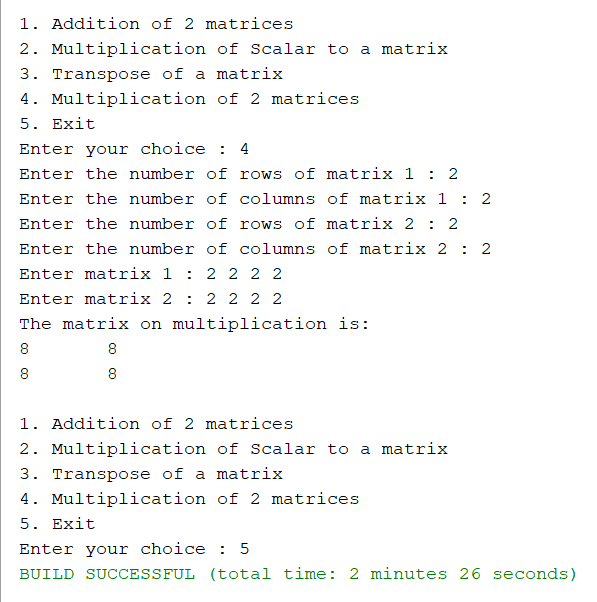
****

**Multiplication of scalar to matrix**

**Transpose of a matrix**

****

**Multiplication of 2 matrices**

****

**CONCLUSION:**

The program to perform various matrix operations in java was implemented successfully.

**Exp No: 03 DATE: 01/12/2021**

**IMPLEMENTATION OF INHERITANCE**

**AIM:**

Write a JAVA program to implement following inheritance: -

**THEORY:**

**Problem Statement:**

A Bank maintains two account for its customers 1) savings account 2) Current account. Saving account provides compound interest and withdrawal facilities but not cheque book facility. The current account provides cheque book facility but no interest.

The current account holder should also maintain a minimum balance of 5000/- and if the balance falls below this level, a service charge is imposed.

Create a class Account that stores customer name,acc no & types of account. From this derive classes curr\_acct and sav\_acct to make them more specific to their requirements.

Include the necessary method in order to achieve the following tasks: -

1. Accept deposit from a customer and update the balance
2. Display the balance
3. Compute and deposit interest
4. Permit withdrawal and update the balance
5. Check for min balance, impose penalty if necessary and update the balance

**Inheritance in Java**

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

**Terms used in Inheritance**

**Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

**Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

**Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

**Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

**The syntax of Java Inheritance**

class Subclass-name extends Superclass-name

{

//methods and fields

}

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

## **Types of inheritance in java**

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical. In java programming, multiple and hybrid inheritance is supported through interface only.





When a class inherits another class, it is known as a single inheritance.

When there is a chain of inheritance, it is known as multilevel inheritance.

When two or more classes inherits a single class, it is known as hierarchical inheritance.

**PROGRAM:**

**OUTPUT:**

**CONCLUSION:**

Program to implement inheritance relation was executed successfully.

**Exp No: 04 DATE: 08/12/2021**

**IMPLEMENTATION OF THREADS USING RUNNABLE INTERFACE**

**AIM:**

Write a Java Program to implement following scenario: -

**THEORY:**

**Problem Statement:**

Let the main method create two Threads by implementing a runnable interface and passes a single object of type Triangle to both the threads. Let both Threads try to simultaneously access draw\_triangle( ) method of triangle object. One thread tries to print triangle of # and other tries to print triangle of @. use synchronisation to achieve the desired result: -

**# @**

**# # # @ @ @**

**# # # # # @ @ @ @ @**

**# # # # # # # @ @ @ @ @ @ @**

**Life cycle of a Thread (Thread States)**

In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

**Explanation of Different Thread States**

**New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

**Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is runnable, and the other is running.

**Blocked or Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

**Timed Waiting:** Sometimes, waiting for leads to starvation. For example, a thread (its name is A) has entered the critical section of a code and is not willing to leave that critical section. In such a scenario, another thread (its name is B) has to wait forever, which leads to starvation. To avoid such scenario, a timed waiting state is given to thread B. Thus, thread lies in the waiting state for a specific span of time, and not forever. A real example of timed waiting is when we invoke the sleep() method on a specific thread. The sleep() method puts the thread in the timed wait state. After the time runs out, the thread wakes up and start its execution from when it has left earlier.

**Terminated:** A thread reaches the termination state because of the following reasons:

* When a thread has finished its job, then it exists or terminates normally.
* Abnormal termination: It occurs when some unusual events such as an unhandled exception or segmentation fault.



**Synchronization in Java**

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Syntax

synchronized(objectidentifier) {

// Access shared variables and other shared resources

}

**Java Runnable Interface**

Java runnable is an interface used to execute code on a concurrent thread. It is an interface which is implemented by any class if we want that the instances of that class should be executed by a thread.

The runnable interface has an undefined method run() with void as return type, and it takes in no arguments.

**public void run()** - This method takes in no arguments. When the object of a class implementing Runnable class is used to create a thread, then the run method is invoked in the thread which executes separately.

**PROGRAM:**

package trianglethread;

class Triangle

{

synchronized void draw\_pyramid(char ch)

{

for(int i=0;i<4;i++)

{

for(int k=0;k<=4-i;k++)

{

System.out.print(" ");

}

for(int j=i\*2;j>=0;j--)

{

System.out.print(ch);

}

System.out.println();

}

}

}

class A implements Runnable

{

Triangle t;

A(Triangle t)

{

this.t=t;

}

public void run()

{

t.draw\_pyramid('#');

}

}

class B implements Runnable

{

Triangle t;

B(Triangle t)

{

this.t=t;

}

public void run()

{

t.draw\_pyramid('@');

}

}

public class TriangleThread

{

public static void main(String[] args)

{

Triangle tobj=new Triangle();

A aObj=new A(tobj);

Thread threadObj=new Thread(aObj);

B bObj=new B(tobj);

Thread threadObj2=new Thread(bObj);

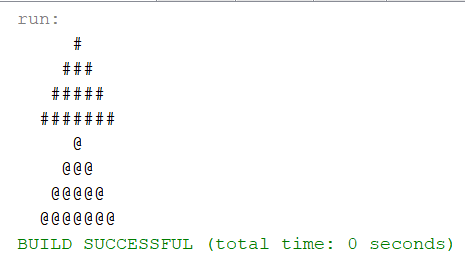
threadObj.start();

threadObj2.start();

}

}

**OUTPUT:**

****

**CONCLUSION:**

Program to implement Threads using runnable interface was executed successfully.

**Exp No: 05 DATE: 28/09/2021**

**IMPLEMENTATION OF ARRAY LIST IN JAVA COLLECTIONS**

**AIM:**

Write a java program to implement array list collection class.

**THEORY:**

**Problem Statement:**

Create console-based menu driven application called Book Store, which have features like add a book, delete a book and search a book.

Create an array list called Shelf, which represent the shelf which contains the book.

Create a class Books which will have 3 attribute names of the book, author and publication.

**Java ArrayList**

Java ArrayList class uses a dynamic array for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the java.util package. It is like the Vector in C++.

The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of List interface here. The ArrayList maintains the insertion order internally.

It inherits the AbstractList class and implements List interface.

The important points about Java ArrayList class are:

* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non synchronized.
* Java ArrayList allows random access because array works at the index basis.
* In ArrayList, manipulation is little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.

### Methods of ArrayList

|  |  |
| --- | --- |
| **Method** | **Description** |
| void [add](https://www.javatpoint.com/java-arraylist-add-method)(int index, E element) | It is used to insert the specified element at the specified position in a list. |
| boolean [add](https://www.javatpoint.com/java-arraylist-add-method)(E e) | It is used to append the specified element at the end of a list. |
| boolean [addAll](https://www.javatpoint.com/java-arraylist-addall-method)(Collection<? extends E> c) | It is used to append 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. |
| boolean [addAll](https://www.javatpoint.com/java-arraylist-addall-method)(int index, Collection<? extends E> c) | It is used to append all the elements in the specified collection, starting at the specified position of the list. |
| void [clear](https://www.javatpoint.com/java-arraylist-clear-method)() | It is used to remove all of the elements from this list. |
| void ensureCapacity(int requiredCapacity) | It is used to enhance the capacity of an ArrayList instance. |
| E get(int index) | It is used to fetch the element from the particular position of the list. |
| boolean isEmpty() | It returns true if the list is empty, otherwise false. |
| int lastIndexOf(Object o) | It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element. |
| Object[] toArray() | It is used to return an array containing all of the elements in this list in the correct order. |
| <T> T[] toArray(T[] a) | It is used to return an array containing all of the elements in this list in the correct order. |
| Object clone() | It is used to return a shallow copy of an ArrayList. |
| boolean contains(Object o) | It returns true if the list contains the specified element |
| int indexOf(Object o) | It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element. |
| E remove(int index) | It is used to remove the element present at the specified position in the list. |
| boolean [remove](https://www.javatpoint.com/java-arraylist-remove-method)(Object o) | It is used to remove the first occurrence of the specified element. |
| boolean [removeAll](https://www.javatpoint.com/java-arraylist-removeall-method)(Collection<?> c) | It is used to remove all the elements from the list. |
| boolean removeIf(Predicate<? super E> filter) | It is used to remove all the elements from the list that satisfies the given predicate. |
| protected void [removeRange](https://www.javatpoint.com/java-arraylist-removerange-method)(int fromIndex, int toIndex) | It is used to remove all the elements lies within the given range. |
| void replaceAll(UnaryOperator<E> operator) | It is used to replace all the elements from the list with the specified element. |
| void [retainAll](https://www.javatpoint.com/java-arraylist-retainall-method)(Collection<?> c) | It is used to retain all the elements in the list that are present in the specified collection. |
| E set(int index, E element) | It is used to replace the specified element in the list, present at the specified position. |
| void sort(Comparator<? super E> c) | It is used to sort the elements of the list on the basis of specified comparator. |
| Spliterator<E> spliterator() | It is used to create spliterator over the elements in a list. |
| List<E> subList(int fromIndex, int toIndex) | It is used to fetch all the elements lies within the given range. |
| int size() | It is used to return the number of elements present in the list. |
| void trimToSize() | It is used to trim the capacity of this ArrayList instance to be the list's current size. |

**PROGRAM:**

**OUTPUT:**

**CONCLUSION:**

Program to implement Array list using collections class in java was executed successfully.

**Exp No: 06 DATE: 28/09/2021**

**IMPLEMENTATION OF HASH TABLE IN JAVA COLLECTIONS**

**AIM:**

Implementation of hash tables class in java collections.

**THEORY:**

**Problem Statement:**

Create a class telephone directory that contains the following fields

telephone number and user name.

Write a menu driven program using the hash table that performs following operations on the telephone directory: -

1) insert a record 2) delete a record 3) search user

**Java HashSet**

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

* HashSet stores the elements by using a mechanism called hashing.
* HashSet contains unique elements only.
* HashSet allows null value.
* HashSet class is non synchronized.
* HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
* HashSet is the best approach for search operations.
* The initial default capacity of HashSet is 16, and the load factor is 0.75.

**Difference between List and Set**

A list can contain duplicate elements whereas Set contains unique elements only.

|  |  |  |  |
| --- | --- | --- | --- |
| **SN** | **Modifier & Type** | **Method** | **Description** |
| 1) | boolean | [add(E e)](https://www.javatpoint.com/java-hashset-add-method) | It is used to add the specified element to this set if it is not already present. |
| 2) | void | [clear()](https://www.javatpoint.com/java-hashset-clear-method) | It is used to remove all of the elements from the set. |
| 3) | object | [clone()](https://www.javatpoint.com/java-hashset-clone-method) | It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned. |
| 4) | boolean | [contains(Object o)](https://www.javatpoint.com/java-hashset-contains-method) | It is used to return true if this set contains the specified element. |
| 5) | boolean | [isEmpty()](https://www.javatpoint.com/java-hashset-isempty-method) | It is used to return true if this set contains no elements. |
| 6) | Iterator<E> | [iterator()](https://www.javatpoint.com/java-hashset-iterator-method) | It is used to return an iterator over the elements in this set. |
| 7) | boolean | [remove(Object o)](https://www.javatpoint.com/java-hashset-remove-method) | It is used to remove the specified element from this set if it is present. |
| 8) | int | [size()](https://www.javatpoint.com/java-hashset-size-method) | It is used to return the number of elements in the set. |
| 9) | Spliterator<E> | [spliterator()](https://www.javatpoint.com/java-hashset-spliterator-method) | It is used to create a late-binding and fail-fast Spliterator over the elements in the set. |

**PROGRAM:**

**OUTPUT:**

**CONCLUSION:**

Program to implement hash table using collections class in java was executed successfully.

**Exp No: 07 DATE: 28/09/2021**

**DESIGNING AWT BASED GUI**

**AIM:**

Implementation of AWT based GUI.

**THEORY:**

**Problem Statement:**

Create frame which has 2 labels username and password, also place 2 textboxes adjacent to them and 2 button login and exit.

Java AWT (Abstract Window Toolkit)

Java AWT is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.

For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

**Components**

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

**Container**

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

**Types of containers:** There are four types of containers in Java AWT:

* Window
* Panel
* Frame
* Dialog

**Window:** The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

**Panel:** The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

**Frame:** The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

**PROGRAM:**

**OUTPUT:**

**CONCLUSION:**

Program to Implement AWT based GUI was executed successfully.