Tasks 2

= = = = =  = =

1.Write 3 different java programs to print the following patterns

                a) 1

                   12

                   123

                   12345

**class** NumberPattern

{

**public** **static** **void** main(String[] args)

{

**for** (**int** i = 1; i <= 5; i++)

{

**for** (**int** j = 1; j <= i; j++)

{

System.***out***.print(j+" ");

}

System.***out***.println();

}

}

}

    b) 54321

        5432

        543

        54

        5

**public** **class** NumPattern2 {

**public** **static** **void** main(String[] args)

{

**for** (**int** i = 1; i <=5; i++)

{

**for** (**int** j = 5; j >=1; j--)

{

System.***out***.print(j+" ");

**if** (j==i)

**break**;

}

System.***out***.println();

}

}

}

            c)     x

                      xxx

                     xxxxx

                   xxxxxxx

                     xxxxx

                      xxx

                       x

              Note: Shape will be Rhombus.

**public** **class** Rhombus {

**public** **static** **void** main(String[] args) {

**int** i,j, k;

**for**(i=1;i<4;i++)

{

**for**(j=i;j<5;j++)

{

System.***out***.print(" ");

}

**for**(k=1;k<(i\*2);k++)

{

System.***out***.print("\*");

}

System.***out***.println();

}

**for**(i=4;i>=1;i--)

{

**for**(j=5;j>i;j--)

{

System.***out***.print(" ");

}

**for**(k=1;k<(i\*2);k++)

{

System.***out***.print("\*");

}

System.***out***.println();

}

}

}

2. Write a java program to take the input from user and determine if it is a prime number or not.

Ans:

**import** java.util.Scanner;

**class** PrimeCheck

{

**public** **static** **void** main(String args[])

{

**boolean** isPrime=**true**;

Scanner scan= **new** Scanner(System.***in***);

System.***out***.println("Enter any number:");

**int** num=scan.nextInt();

**for**(**int** i=2;i<=num/2;i++)

{

**if**(num%i==0)

{

isPrime=**false**;

**break**;

}

}

**if**(isPrime)

System.***out***.println(num + " is a Prime Number");

**else**

System.***out***.println(num + " is not a Prime Number");

}

}

3. Write a java program to display the fibonacci series till less than 200 using only 2 variables.

**public** **class** Fibonacci {

**public** **static** **void** main(String[] args) {

**int** a = 0, b = 1;

System.***out***.println(a);

System.***out***.println(b);

**while**(a+b<200) {

System.***out***.println(a+b);

b = a + b;

a = b - a;

}

}

}

5.Write Java program to check if a name is palindrome.

**import** java.util.Scanner;

**public** **class** PalindromeCheck {

**public** **static** **void** isPalindrome(String s)

{

String reverse = **new** StringBuffer(s).reverse().toString();

// check whether the string is palindrome or not

**if** (s.equals(reverse))

System.***out***.println("Palindrome");

**else**

System.***out***.println("Not Palindrome");

}

**public** **static** **void** main (String[] args)

{

String str, rev = "";

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter a string:");

str = sc.nextLine();

*isPalindrome*(str);

}

}

6.Write Java program to check if a number is Armstrong number or not? (input 153 output true,  123 output false)

**import** java.util.Scanner;

**public** **class** ArmStrongTest {

**public** **static** **void** main(String args[]) {

System.***out***.println("Please enter a 3 digit number to find if its an Armstrong number:");

**int** number = **new** Scanner(System.***in***).nextInt();

**if**(*isArmStrong*(number)){

System.***out***.println("Number : " + number + " is an Armstrong number");

}

**else**{

System.***out***.println("Number : " + number + " is not an Armstrong number");

}

}

**private** **static** **boolean** isArmStrong(**int** number)

{

**int** result = 0;

**int** orig = number;

**while**(number != 0)

{

**int** r = number%10;

result = result + r\*r\*r;

number = number/10;

}

**if**(orig == result)

{

**return** **true**;

}

**return** **false**;

}

}

7.How to find factorial of number in Java using iteration?

**import** java.util.Scanner;

**public** **class** Factorial {

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter the number whose factorial is to be found: ");

**int** n = scanner.nextInt();

**int** result = *factorial*(n);

System.***out***.println("The factorial of " + n + " is " + result);

}

**public** **static** **int** factorial(**int** n) {

**int** result = 1;

**for** (**int** i = 1; i <= n; i++) {

result = result \* i;

}

**return** result;

}

}

8.Write a Java code to take a character as a input from user and determine if it is a vowel or a consonant using conditional construct.

**import** java.util.Scanner;

**public** **class** CheckChar {

**public** **static** **void** main(String[ ] arg)

{

**int** i=0;

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter a character : ");

**char** ch=sc.next( ).charAt(0);

**if**(ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u'||ch=='A'||ch=='E'||ch=='I'||ch=='O'||ch=='U')

{

System.***out***.println("Entered character "+ch+" is Vowel");

}

**else** **if**((ch>='a'&&ch<='z')||(ch>='A'&&ch<='Z'))

System.***out***.println("Entered character "+ch+" is Consonant");

**else**

System.***out***.println("Not an alphabet");

}

}

9. Write a switch case java code to create calculator with + - / \* functionalities only.

**package** com.niit;

**import** java.util.Scanner;

**public** **class** Calculator {

**public** **static** **void** main(String args[])

{

Scanner sc=**new** Scanner(System.***in***);

**int** a,b;

**char** character;

System.***out***.print("enter ur firt value : ");

a=sc.nextInt();

System.***out***.print("enter ur second value : ");

b=sc.nextInt();

System.***out***.print("enter ur operator +,-,\*,/ : ");

character=sc.next().charAt(0);

**switch**(character){

**case** '-':

System.***out***.print(a-b);

**break**;

**case** '+':

System.***out***.print(a+b);

**break**;

**case** '\*':

System.***out***.print(a\*b);

**break**;

**case** '/':

System.***out***.print(a/b);

**break**;

**default** :

System.***out***.print("syntax error");

**break**;

}

}

}

10. Write a java code to copy one array into another.

**import** java.util.Arrays;

**public** **class** CopyArray {

**public** **static** **void** main(String args[]){

**int** a[]={10,20,30,40,50};

**int** b[]=**new** **int**[a.length];

//copying one array to another

b=Arrays.*copyOf*(a,a.length);

//printing array

**for**(**int** i=0;i<b.length;++i){

System.***out***.print(b[i]+" ");

}

}

}

public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

**public** **class** CopyArray{

**public** **static** **void** main(String args[]){

**int** a[]={10,20,30,40,50};

**int** b[]=**new** **int**[a.length];

//copying one array to another

System.*arraycopy*(a,0,b,0,a.length);

//printing array

**for**(**int** i=0;i<b.length;++i){

System.***out***.print(b[i]+" ");

}

}

}

* System.arraycopy() uses JNI (Java Native Interface) to copy an array (or parts of it), so it isfast.
* clone() creates a **new** array with the same characteristics as the old array, i.e., same size, same type, and *same* contents.
* [Arrays.copyOf()](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html#copyOf%28T%5b%5d,%20int%29) can be used to create a copy of another array with a different size.

Arrays.copyOf(..) uses System.arrayCopy(..) method internally.

11. Write a java code to compare the length of two arrays and display the longer array.

**public** **class** ArrayLengthCompare {

**public** **static** **void** main(String a[]){

**int**[] arr1 = {4,7,3,9,2,7,8,56,89};

**int**[] arr2 = {3,2,12,9,40,32,4};

**if**(arr1.length>arr2.length) {

**for**(**int** x:arr1)

System.***out***.println(x);

}

**else** {

**for**(**int** x:arr2)

System.***out***.println(x);

}

}

}

12. Write a java code to display a reverse String array.

**import** java.util.Collections;

**import** java.util.List;

**import** java.util.Arrays;

**public** **class** ReverseStringArray {

**public** **static** **void** main(String args[]){

//String array

String[] strDays = **new** String[]{"Sunday", "Monday", "Tuesday", "Wednesday"};

//first create a list from String array

List<String> list = Arrays.*asList*(strDays);

//next, reverse the list using Collections.reverse method

Collections.*reverse*(list);

//next, convert the list back to String array

strDays = (String[]) list.toArray();

System.***out***.println("String array reversed");

//print the reversed String array

**for**(**int** i=0; i < strDays.length; i++){

System.***out***.println(strDays[i]);

}

}

13.   Write the difference between checked and unchecked exception with example code

|  |  |
| --- | --- |
| **Checked Exceptions** | **Unchecked Exceptions** |
| They are known at compile time. | They are known at run time. |
| They are checked at compile time. | They are not checked at compile time. Because they occur only at run time. |
| These are compile time exceptions. | These are run time exceptions. |
| If  these exceptions are not handled properly in the application, they give compile time error. | If these exceptions are not handled properly, they don’t give compile time error. But application will be terminated prematurely at run time. |
| All sub classes of java.lang.Exception Class except sub classes of RunTimeException are checked exceptions. | All sub classes of RunTimeException and sub classes of java.lang.Error are unchecked exceptions. |

Here are the few Checked Exceptions –

* SQLException
* IOException
* ClassNotFoundException
* InvocationTargetException

Example

import java.io.\*;

class Example {

public static void main(String args[])

{

FileInputStream fis = null;

/\*This constructor FileInputStream(File filename)

\* throws FileNotFoundException which is a checked

\* exception

\*/

fis = new FileInputStream("B:/myfile.txt");

int k;

/\* Method read() of FileInputStream class also throws

\* a checked exception: IOException

\*/

while(( k = fis.read() ) != -1)

{

System.out.print((char)k);

}

/\*The method close() closes the file input stream

\* It throws IOException\*/

fis.close();

}

}

Here are the few unchecked exception classes:

* NullPointerException
* ArrayIndexOutOfBoundsException
* ArithmeticException
* IllegalArgumentException
* NumberFormatException

Example:

class Example {

public static void main(String args[])

{

int num1=10;

int num2=0;

/\*Since I'm dividing an integer with 0

\* it should throw ArithmeticException

\*/

int res=num1/num2;

System.out.println(res);

}

}

14.   Write the difference between throw and throws with example code

|  |  |  |
| --- | --- | --- |
| **Sl.No** | **throw** | **throws** |
| 1) | Java throw keyword is used to explicitly throw an exception. | Java throws keyword is used to declare an exception. |
| 2) | Checked exception cannot be propagated using throw only. | Checked exception can be propagated with throws. |
| 3) | Throw is followed by an instance. | Throws is followed by class. |
| 4) | Throw is used within the method. | Throws is used with the method signature. |
| 5) | You cannot throw multiple exceptions. | You can declare multiple exceptions e.g. public void method()throws IOException,SQLException. |

For example:  
**Throw:**

...

void myMethod() {

try {

//throwing arithmetic exception using throw

throw new ArithmeticException("Something went wrong!!");

}

catch (Exception exp) {

System.out.println("Error: "+exp.getMessage());

}

}

...

**Throws:**

...

//Declaring arithmetic exception using throws

void sample() throws ArithmeticException{

//Statements

}

...

15.   Write a note or nested try…catch block with example code

When a [try catch block](https://beginnersbook.com/2013/04/try-catch-in-java/) is present in another try block then it is called the nested try catch block. Each time a try block does not have a catch handler for a particular [exception](https://beginnersbook.com/2013/04/java-exception-handling/), then the catch blocks of parent try block are inspected for that exception, if match is found that that catch block executes.

If neither catch block nor parent catch block handles exception then the system generated message would be shown for the exception, similar to what we see when we don’t handle exception.

**Syntax of Nested try Catch**

....

//Main try block

try {

statement 1;

statement 2;

//try-catch block inside another try block

try {

statement 3;

statement 4;

//try-catch block inside nested try block

try {

statement 5;

statement 6;

}

catch(Exception e2) {

//Exception Message

}

}

catch(Exception e1) {

//Exception Message

}

}

//Catch of Main(parent) try block

catch(Exception e3) {

//Exception Message

}

....

**Nested Try Catch Example**

Here we have deep (two level) nesting which means we have a try-catch block inside a nested try block. To make you understand better I have given the names to each try block in comments like try-block2, try-block3 etc.

This is how the structure is: try-block3 is inside try-block2 and try-block2 is inside main try-block, you can say that the main try-block is a grand parent of the try-block3. Refer the explanation which is given at the end of this code.

class NestingDemo{

public static void main(String args[]){

//main try-block

try{

//try-block2

try{

//try-block3

try{

int arr[]= {1,2,3,4};

/\* I'm trying to display the value of

\* an element which doesn't exist. The

\* code should throw an exception

\*/

System.out.println(arr[10]);

}catch(ArithmeticException e){

System.out.print("Arithmetic Exception");

System.out.println(" handled in try-block3");

}

}

catch(ArithmeticException e){

System.out.print("Arithmetic Exception");

System.out.println(" handled in try-block2");

}

}

catch(ArithmeticException e3){

System.out.print("Arithmetic Exception");

System.out.println(" handled in main try-block");

}

catch(ArrayIndexOutOfBoundsException e4){

System.out.print("ArrayIndexOutOfBoundsException");

System.out.println(" handled in main try-block");

}

catch(Exception e5){

System.out.print("Exception");

System.out.println(" handled in main try-block");

}

}

}

16.   Write a note on MultiThreading and MultiTasking

1. The basic difference between multitasking and multithreading is that in **multitasking**, the system allows executing multiple programs and tasks at the same time, whereas, in **multithreading**, the system executes multiple threads of the same or different processes at the same time.
2. In multitasking **CPU** has to **switch** between **multiple programs** so that it appears that multiple programs are running simultaneously. On other hands, in multithreading **CPU** has to **switch** between **multiple threads** to make it appear that all threads are running simultaneously.
3. Multitasking allocates**separate memory and resources** for each process/program whereas, in multithreading threads belonging to the same process**shares the same memory and resources** as that of the process.

Multitasking is similar to multiprogramming whereas, Multithreading is thread-based multitasking. Multithreading is is less costlier than multitaskings as threads are easy to create then a process.

17.   Write a short note on Deque and give example code.

The java.util.Deque interface is a subtype of the [java.util.Queue](https://www.geeksforgeeks.org/queue-interface-java/) interface. The Deque is related to the double-ended queue that supports addition or removal of elements from either end of the data structure, it can be used as a [queue (first-in-first-out/FIFO)](https://www.geeksforgeeks.org/queue/) or as a [stack (last-in-first-out/LIFO)](https://www.geeksforgeeks.org/stack/).

**Methods** of deque:

1. **add(element):**Adds an element to the tail.
2. **addFirst(element):**Adds an element to the head.
3. **addLast(element):** Adds an element to the tail.
4. **offer(element):**Adds an element to the tail and returns a boolean to explain if the insertion was successful.
5. **offerFirst(element):**Adds an element to the head and returns a boolean to explain if the insertion was successful.
6. **offerLast(element):**Adds an element to the tail and returns a boolean to explain if the insertion was successful.
7. **iterator():** Returna an iterator for this deque.
8. **descendingIterator():** Returns an iterator that has the reverse order for this deque.
9. **push(element):**Adds an element to the head.
10. **pop(element):** Removes an element from the head and returns it.
11. **removeFirst():**Removes the element at the head.
12. **removeLast():** Removes the element at the tail.

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | | Throws an exception if operation fails. | Returns null or false if operation fails. |
| Insertion | Front End | addFirst() | offerFirst() |
| Rear End | addLast() | offerLast() |
| Retrieval | Front End | getFirst() | peekFirst() |
| Rear End | getLast() | peekLast() |
| Retrieval And Removal | Front End | removeFirst() | pollFirst() |
| Rear End | removeLast() | pollLast() |

**import** java.util.Deque;

**import** java.util.Iterator;

**import** java.util.LinkedList;

**public** **class** DequeueExample {

**public** **static** **void** main(String[] args) {

Deque deque = **new** LinkedList<>();

// We can add elements to the queue in various ways

deque.add("Element 1 (Tail)"); // add to tail

deque.addFirst("Element 2 (Head)");

deque.addLast("Element 3 (Tail)");

deque.push("Element 4 (Head)"); //add to head

deque.offer("Element 5 (Tail)");

deque.offerFirst("Element 6 (Head)");

deque.offerLast("Element 7 (Tail)");

System.***out***.println(deque + "\n");

// Iterate through the queue elements.

System.***out***.println("Standard Iterator");

Iterator iterator = deque.iterator();

**while** (iterator.hasNext()) {

System.***out***.println("\t" + iterator.next());

}

// Reverse order iterator

Iterator reverse = deque.descendingIterator();

System.***out***.println("Reverse Iterator");

**while** (reverse.hasNext()) {

System.***out***.println("\t" + reverse.next());

}

// Peek returns the head, without deleting it from the deque

System.***out***.println("Peek " + deque.peek());

System.***out***.println("After peek: " + deque);

// Pop returns the head, and removes it from the deque

System.***out***.println("Pop " + deque.pop());

System.***out***.println("After pop: " + deque);

// We can check if a specific element exists in the deque

System.***out***.println("Contains element 3: " + deque.contains("Element 3 (Tail)"));

// We can remove the first / last element.

deque.removeFirst();

deque.removeLast();

System.***out***.println("Deque after removing first and last: " + deque);

}

}

18.   Write a short note on Generics an all types of Parameters used in Generics with example code.

Generics was added in Java 5 to provide **compile-time type checking** and removing risk ofClassCastException that was common while working with collection classes. The whole collection framework was re-written to use generics for type-safety

we use <> to specify parameter types in generic class creation. To create objects of generic class, we use following syntax.

// To create an instance of generic class

BaseType <Type> obj = new BaseType <Type>()

**Note:** In Parameter type we can not use primitives like

'int','char' or 'double'.

|  |
| --- |
| // A Simple Java program to show working of user defined  // Generic classes    // We use < > to specify Parameter type  class Test<T>  {      // An object of type T is declared      T obj;      Test(T obj) {  this.obj = obj;  }  // constructor      public T getObject()  { return this.obj; }  }    // Driver class to test above  class Main  {      public static void main (String[] args)      {          // instance of Integer type          Test <Integer> iObj = new Test<Integer>(15);          System.out.println(iObj.getObject());            // instance of String type          Test <String> sObj =                            new Test<String>("Helloworld");          System.out.println(sObj.getObject());      }  } |

We can also pass multiple Type parameters in Generic classes.

|  |
| --- |
| // A Simple Java program to show multiple  // type parameters in Java Generics    // We use < > to specify Parameter type  class Test<T, U>  {      T obj1;  // An object of type T      U obj2;  // An object of type U        // constructor      Test(T obj1, U obj2)      {          this.obj1 = obj1;          this.obj2 = obj2;      }        // To print objects of T and U      public void print()      {          System.out.println(obj1);          System.out.println(obj2);      }  }    // Driver class to test above  class Main  {      public static void main (String[] args)      {          Test <String, Integer> obj =              new Test<String, Integer>("GfG", 15);            obj.print();      }  } |

**Generic Functions:**

We can also write generic functions that can be called with different types of arguments based on the type of arguments passed to generic method, the compiler handles each method.

|  |
| --- |
| // A Simple Java program to show working of user defined  // Generic functions    class Test  {      // A Generic method example      static <T> void genericDisplay (T element)      {          System.out.println(element.getClass().getName() +                             " = " + element);      }        // Driver method      public static void main(String[] args)      {           // Calling generic method with Integer argument          genericDisplay(11);            // Calling generic method with String argument          genericDisplay("HelloWorld");            // Calling generic method with double argument          genericDisplay(1.0);      }  } |

19.   Write a short note on Map Interface.

A [Map](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html) is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value. It models the mathematical *function* abstraction. TheMap interface includes methods for basic operations (such as put, get, remove, containsKey, containsValue, size, and empty), bulk operations (such as putAll andclear), and collection views (such as keySet, entrySet, and values).

The Java platform contains three general-purpose Map implementations: [HashMap](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html" \t "_blank), [TreeMap](https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html" \t "_blank), and [LinkedHashMap](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html" \t "_blank)

## Methods Of Map Interface In Java :

|  |  |  |
| --- | --- | --- |
| SL NO. | Methods | Descriptions |
| 1 | int size() | Returns number of key-value pairs in this map. |
| 2 | boolean isEmpty() | Checks whether this map is empty or not. |
| 3 | boolean containsKey(Object key) | Returns true if this map contains a mapping for the specified key. |
| 4 | boolean containsValue(Object value) | Returns true if this map contains one or more keys associated with the specified value. |
| 5 | V get(Object key) | Returns value associated with the specified key. |
| 6 | V put(K key, V value) | Adds the specified key-value pair to this map. If the specified key already exist in the map, old value will be replaced by the specified value. |
| 7 | V remove(Object key) | Removes the specified key along with it’s value from this map. |
| 8 | void putAll(Map<? extends K, ? extends V> m) | Copies all key-value pairs from the specified map to this map. |
| 9 | void clear() | Removes all mappings from this map. |
| 10 | Set<K> keySet() | Returns a set containing all keys of this map. The returned set is backed by actual map. So, changes made to the map are reflected in the set and vice-versa. |
| 11 | Collection<V> values() | Returns a collection of values of this map. The returned collection is backed by actual map. So, any changes made to the map is reflected in collection and vice-versa. |
| 12 | Set<Map.Entry<K, V>> entrySet() | Returns set view of the mappings contained in this map. |
| 13 | boolean equals(Object o) | Compares the specified object with this map. |
| 14 | int hashCode() | Returns hashcode value of this map. |

## Properties Of Map Interface In Java :

1) Map interface is a part of Java Collection Framework, but it doesn’t inherit **Collection Interface**.

2) Map interface stores the data as a **key-value pairs** where each key is associated with a value.

3) A map can not have duplicate **keys** but can have duplicate **values**.

4) Each key **at most** must be associated with one value.

5) Each key-value pairs of the map are stored as **Map.Entry** objects. Map.Entry is an inner interface of Map interface.

6) The common implementations of Map interface are **HashMap**, **LinkedHashMap** and **TreeMap**.

7) Order of elements in map is implementation dependent. **HashMap** doesn’t maintain any order of elements. **LinkedHashMap** maintains **insertion order** of elements. Where as **TreeMap** places the elements according to supplied **Comparator**.

8) The Map interface provides three methods, which allows map’s contents to be viewed as a **set of keys**(keySet() method), **collection of values**(values() method), or **set of key-value mappings**(entrySet() method).

20.   Write the difference between LinkedList and ArrayList.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses **dynamic array** to store the elements. | LinkedList internally uses **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory. |
| 3) ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |

21.   Write a note on Dynamic array in java.

Vector implements a dynamic array. It is similar to ArrayList, but with two differences −

* Vector is synchronized.
* Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.

Following is the list of constructors provided by the vector class.

|  |  |
| --- | --- |
| **Sr.No.** | **Constructor & Description** |
| 1 | **Vector( )**  This constructor creates a default vector, which has an initial size of 10. |
| 2 | **Vector(int size)**  This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size. |
| 3 | **Vector(int size, int incr)**  This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward. |
| 4 | **Vector(Collection c)**  This constructor creates a vector that contains the elements of collection c. |

Apart from the methods inherited from its parent classes, Vector defines the following methods −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **void add(int index, Object element)**  Inserts the specified element at the specified position in this Vector. |
| 2 | **boolean add(Object o)**  Appends the specified element to the end of this Vector. |
| 3 | **boolean addAll(Collection c)**  Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection's Iterator. |
| 4 | **boolean addAll(int index, Collection c)**  Inserts all of the elements in in the specified Collection into this Vector at the specified position. |
| 5 | **void addElement(Object obj)**  Adds the specified component to the end of this vector, increasing its size by one. |
| 6 | **int capacity()**  Returns the current capacity of this vector. |
| 7 | **void clear()**  Removes all of the elements from this vector. |
| 8 | **Object clone()**  Returns a clone of this vector. |
| 9 | **boolean contains(Object elem)**  Tests if the specified object is a component in this vector. |
| 10 | **boolean containsAll(Collection c)**  Returns true if this vector contains all of the elements in the specified Collection. |
| 11 | **void copyInto(Object[] anArray)**  Copies the components of this vector into the specified array. |
| 12 | **Object elementAt(int index)**  Returns the component at the specified index. |
| 13 | **Enumeration elements()**  Returns an enumeration of the components of this vector. |
| 14 | **void ensureCapacity(int minCapacity)**  Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument. |
| 15 | **boolean equals(Object o)**  Compares the specified Object with this vector for equality. |
| 16 | **Object firstElement()**  Returns the first component (the item at index 0) of this vector. |
| 17 | **Object get(int index)**  Returns the element at the specified position in this vector. |
| 18 | **int hashCode()**  Returns the hash code value for this vector. |
| 19 | **int indexOf(Object elem)**  Searches for the first occurence of the given argument, testing for equality using the equals method. |
| 20 | **int indexOf(Object elem, int index)**  Searches for the first occurence of the given argument, beginning the search at index, and testing for equality using the equals method. |
| 21 | **void insertElementAt(Object obj, int index)**  Inserts the specified object as a component in this vector at the specified index. |
| 22 | **boolean isEmpty()**  Tests if this vector has no components. |
| 23 | **Object lastElement()**  Returns the last component of the vector. |
| 24 | **int lastIndexOf(Object elem)**  Returns the index of the last occurrence of the specified object in this vector. |
| 25 | **int lastIndexOf(Object elem, int index)**  Searches backwards for the specified object, starting from the specified index, and returns an index to it. |
| 26 | **Object remove(int index)**  Removes the element at the specified position in this vector. |
| 27 | **boolean remove(Object o)**  Removes the first occurrence of the specified element in this vector, If the vector does not contain the element, it is unchanged. |
| 28 | **boolean removeAll(Collection c)**  Removes from this vector all of its elements that are contained in the specified Collection. |
| 29 | **void removeAllElements()**  Removes all components from this vector and sets its size to zero. |
| 30 | **boolean removeElement(Object obj)**  Removes the first (lowest-indexed) occurrence of the argument from this vector. |
| 31 | **void removeElementAt(int index)**  removeElementAt(int index). |
| 32 | **protected void removeRange(int fromIndex, int toIndex)**  Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive. |
| 33 | **boolean retainAll(Collection c)**  Retains only the elements in this vector that are contained in the specified Collection. |
| 34 | **Object set(int index, Object element)**  Replaces the element at the specified position in this vector with the specified element. |
| 35 | **void setElementAt(Object obj, int index)**  Sets the component at the specified index of this vector to be the specified object. |
| 36 | **void setSize(int newSize)**  Sets the size of this vector. |
| 37 | **int size()**  Returns the number of components in this vector. |
| 38 | **List subList(int fromIndex, int toIndex)**  Returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive. |
| 39 | **Object[] toArray()**  Returns an array containing all of the elements in this vector in the correct order. |
| 40 | **Object[] toArray(Object[] a)**  Returns an array containing all of the elements in this vector in the correct order; the runtime type of the returned array is that of the specified array. |
| 41 | **String toString()**  Returns a string representation of this vector, containing the String representation of each element. |
| 42 | **void trimToSize()**  Trims the capacity of this vector to be the vector's current size. |

22.   What is the purpose of the System class?

1.System class is provided with useful fields (static members) pertaining to the environment.  
  
2. Standard input,output and error output streams are provided with System class. These are used to access the externally defined properties and environment variables.  
  
**Example :**

System.in - external property for input device.  
System.out – external property for output device

3. Other useful methods that interact with external system / environment are:  
  
- **currentTimeMillis()** – returns the current time in milliseconds  
- **exit()** - terminates currently running JVM  
- **gc()** - invokes the garbage collector  
- **getProperties()** - returns the system properties.  
  
4. The System class can not be instantiated.

23.   Which is the abstract parent class of FileWriter ?

  OutputStreamWriter

24.   Which class is used to read streams of characters from a file?

FileReader

25.   Which class is used to read streams of raw bytes from a file?

FileInputStream

26.   What are the differences between FileInputStream/FileOutputStream and RandomAccessFile

RandomAccessFile treats the file as an array of bytes where it has the internal pointer. The fact that it treats it like a large array of bytes is what is unique about this class. FileInputStream however just reads the stream and returns the data. It is more suited to reading raw data like images etc. It does not treat the file as a large array, it just keeps tabs of where in the file it has read so far. With FileInputStream you would actually have to read the data and place it into an array to get the same style of access as RandomAccessFile.

27.   Write a note on Channels and Buffer with example.

# **Java NIO Channels**

In Java NIO, the channel is a medium used to transports the data efficiently between the entity and byte buffers. It reads the data from an entity and places it inside buffer blocks for consumption.

Channels act as gateway provided by java NIO to access the I/O mechanism. Usually channels have one-to-one relationship with operating system file descriptor for providing the platform independence operational feature.

# **Java NIO Buffers**

Buffers are defined inside **java.nio** package. It defines the core functionality which is common to all buffers: limit, capacity and current position.

Java NIO buffers are used for interacting with NIO channels. It is the block of memory into which we can write data, which we can later be read again. The memory block is wrapped with a NIO buffer object, which provides easier methods to work with the memory block.

28.   What is the difference between System.out ,System.err and System.in?

**System.in**

System.in is an **[InputStream](http://tutorials.jenkov.com/java-io/inputstream.html)** which is typically connected to keyboard input of console programs. System.in is not used as often since data is commonly passed to a command line Java application via command line arguments, or configuration files. In applications with GUI the input to the application is given via the GUI. This is a separate input mechanism from Java IO.

**System.out**

System.out is a **[PrintStream](http://tutorials.jenkov.com/java-io/printstream.html)**. System.out normally outputs the data you write to it to the console. This is often used from console-only programs like command line tools. This is also often used to print debug statements of from a program (though it may arguably not be the best way to get debug info out of a program).

**System.err**

System.err is a **[PrintStream](http://tutorials.jenkov.com/java-io/printstream.html)**. System.err works like System.out except it is normally only used to output error texts. Some programs (like Eclipse) will show the output to System.err in red text, to make it more obvious that it is error text.

35.   Write a note on PreparedStatement and ResultSetMetaData interfaces with code snippets.

# **PreparedStatement interface**

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

1. String sql="insert into emp values(?,?,?)";

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

### **Why use PreparedStatement?**

**Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

#### **How to get the instance of PreparedStatement?**

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

1. **public** PreparedStatement prepareStatement(String query)**throws** SQLException{}

### **Methods of PreparedStatement interface**

The important methods of PreparedStatement interface are given below:

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void setInt(int paramIndex, int value) | sets the integer value to the given parameter index. |
| public void setString(int paramIndex, String value) | sets the String value to the given parameter index. |
| public void setFloat(int paramIndex, float value) | sets the float value to the given parameter index. |
| public void setDouble(int paramIndex, double value) | sets the double value to the given parameter index. |
| public int executeUpdate() | executes the query. It is used for create, drop, insert, update, delete etc. |
| public ResultSet executeQuery() | executes the select query. It returns an instance of ResultSet. |

1. PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");

# **Java ResultSetMetaData Interface**

The metadata means data about data i.e. we can get further information from the data.

If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

## Commonly used methods of ResultSetMetaData interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| public int getColumnCount()throws SQLException | it returns the total number of columns in the ResultSet object. |
| public String getColumnName(int index)throws SQLException | it returns the column name of the specified column index. |
| public String getColumnTypeName(int index)throws SQLException | it returns the column type name for the specified index. |
| public String getTableName(int index)throws SQLException | it returns the table name for the specified column index. |

ResultSetMetaData rsmd=rs.getMetaData();

System.out.println("Total columns: "+rsmd.getColumnCount());

System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));

System.out.println("Column Type Name of 1st column: "+rsmd.getColumnTypeName(1);

36.   Write a note on DDL, DML, DQL, DDL with code snippets.

1. **DDL(Data Definition Language) :**DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database.

**Examples of DDL commands:**

* + [**CREATE**](https://www.geeksforgeeks.org/sql-create/) – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
  + [**DROP**](https://www.geeksforgeeks.org/sql-drop-truncate/) – is used to delete objects from the database.
  + [**ALTER**](https://www.geeksforgeeks.org/sql-alter-add-drop-modify/)-is used to alter the structure of the database.
  + [**TRUNCATE**](https://www.geeksforgeeks.org/sql-drop-truncate/)–is used to remove all records from a table, including all spaces allocated for the records are removed.
  + [**COMMENT**](https://www.geeksforgeeks.org/sql-comments/) –is used to add comments to the data dictionary.
  + [**RENAME**](https://www.geeksforgeeks.org/sql-alter-rename/)–is used to rename an object existing in the database.

1. **DML(Data Manipulation Language) :**The SQL commands that deals with the manipulation of data present in database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

**Examples of DML:**

* + [**SELECT**](https://www.geeksforgeeks.org/sql-select-clause/) – is used to retrieve data from the a database.
  + [**INSERT**](https://www.geeksforgeeks.org/sql-insert-statement/) – is used to insert data into a table.
  + [**UPDATE**](https://www.geeksforgeeks.org/sql-update-statement/) – is used to update existing data within a table.
  + [**DELETE**](https://www.geeksforgeeks.org/sql-delete-statement/) – is used to delete records from a database table.

1. **DCL(Data Control Language) :**DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

**Examples of DCL commands:**

* + **GRANT**-gives user’s access privileges to database.
  + **REVOKE**-withdraw user’s access privileges given by using the GRANT command.

1. **TCL(transaction Control Language) :**TCL commands deals with the [transaction within the database](https://www.geeksforgeeks.org/sql-transactions/).

**Examples of TCL commands:**

* + **COMMIT**– commits a Transaction.
  + [**ROLLBACK**](https://www.geeksforgeeks.org/sql-transactions/)– rollbacks a transaction in case of any error occurs.
  + **SAVEPOINT**–sets a savepoint within a transaction.
  + **SET TRANSACTION**–specify characteristics for the transaction.

37.   Write a note on HTML , CSS and Javascript.

## What is HTML?

HTML is the standard markup language for creating Web pages.

* HTML stands for Hyper Text Markup Language
* HTML describes the structure of Web pages using markup
* HTML elements are the building blocks of HTML pages
* HTML elements are represented by tags
* HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
* Browsers do not display the HTML tags, but use them to render the content of the page

## What is CSS?

* **CSS** stands for **C**ascading **S**tyle **S**heets
* CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**
* CSS **saves a lot of work**. It can control the layout of multiple web pages all at once
* External stylesheets are stored in **CSS files**

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

38.   Write a code to fetch the data from H2 and put it in any collection object and display it.

**public** **class** Student {

**int** id;

String name;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

@Override

**public** String toString() {

**return** "Student [id=" + id + ", name=" + name + "]";

}

}

**import** java.sql.\*;

**import** java.util.ArrayList;

**public** **class** H2Test {

**public** **static** **void** main(String[] args) {

String url = "jdbc:h2:tcp://localhost/~/test";

String user = "sa";

String pwd = "";

String query = "Select \* from Student";

ArrayList<Student> Students = **new** ArrayList<>();

**try** (Connection con = DriverManager.*getConnection*(url, user, pwd);

Statement st = con.createStatement();

ResultSet rs = st.executeQuery(query)) {

**while** (rs.next()) {

Student stud = **new** Student();

stud.setId(rs.getInt(1));

stud.setName(rs.getString(2));

Students.add(stud);

}

}

**catch**(

SQLException e)

{

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**for**(Student s:Students)

System.***out***.println(s);

// **TODO** Auto-generated method stub

}

}

39.   Describe the different approaches of String processing.

String class: When you performed string concatenation in JDK version 1.1, the String class created three string objects in the heap. Of the three string objects, two object references were lost and was garbage collected. If the number of strings for concatenation increased by 10, 20, and so on, intermediate string references were created, and the heap memory was occupied for garbage collection. To solve the problem, JDK version 1.3 introduced the StringBuffer class.

StringBuffer class: You use the StringBuffer class for multithreaded applications. Every string buffer has a capacity. As long as the length of the character sequence in the string buffer doesn't exceed the capacity of 2^31-1, you don't have to allocate a new internal buffer array. If you increase the maximum number of elements in the array, the array goes to negative and throws an error. String buffers are safe to use when multiple threads are involved in the application. In the JDK 5 release, the StringBuffer class was supplemented with an equivalent class,StringBuilder, which was designed for single-thread applications.

StringBuilder class: In general you should use the StringBuilder class instead of the StringBuffer class. The StringBuilder class executes faster, uses memory more efficiently, and provides better performance when you must make many concatenations. The  StringBuilderclass has no array size limit and supports all StringBuffer class methods. When you perform string concatenation with the + operator, Java internally converts that call to the corresponding StringBuilder append() method.