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

                a) 1

                   12

                   123

                   12345

Ans:

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

Ans:

for(int i = 5; i >= 1; i--){

int num = 5;

for(int j = i; j >= 1; j--){

System.out.print(num);

num --;

}

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:

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

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

**if**(num % 2 == 0)

System.***out***.println(num + " is prime");

**else**

System.***out***.println(num + " is not prime");

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

Ans:

**int** number = 0;

**int** add = 0;

**while** (number < 200) {

**if** (add == 0) {

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

add++;

} **else** {

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

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

}

add += number;

number += add;

}

**4. Write Java program to check if a name is palindrome.**

Ans:

Import java.util.Scanner;

String str, rev = "";

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

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

str = sc.nextLine();

**int** length = str.length();

**for** (**int** i = length - 1; i >= 0; i--)

rev = rev + str.charAt(i);

**if** (str.equals(rev))

System.***out***.println(str + " is a palindrome");

**else**

System.***out***.println(str + " is not a palindrome");

**5. Write Java program to check if a number is Armstrong number or not?**

Ans:

int c = 0, a, temp;

int n = 153;// It is the number to check armstrong

temp = n;

while (n > 0) {

a = n % 10;

n = n / 10;

c = c + (a \* a \* a);

}

if (temp == c)

System.out.println("armstrong number");

else

System.out.println("Not armstrong number");

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

Ans:

int i,fact=1;

int number=5;//It is the number to calculate factorial

for(i=1; i<=number; i++){

fact=fact\*i;

}

System.out.println("Factorial of "+number+" is: "+fact);

**7. 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.**

Ans:

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");

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

Ans:

Scanner s=new Scanner(System.in);

        System.out.println("YOU HAVE FOLLOWING CHOICES : ");

        System.out.println("1. ADDITION");

        System.out.println("2. SUBTRACTION ");

        System.out.println("3. MULTIPLICATION ");

        System.out.println("4. DIVISION");

        System.out.println("ENTER YOUR CHOICE : ");

        int i=s.nextInt();

        System.out.println("ENTER FIRST NUMBER ");

        int a=s.nextInt();

        System.out.println("ENTER SECOND NUMBER ");

        int b=s.nextInt();

        double result=0;//'result' will store the result of operation

        switch(i)

        {

            case 1:

                result=a+b;

                break;

            case 2:

                result=a-b;

                break;

            case 3:

                result=a\*b;

                break;

            case 4:

                if(b==0)//when denominator becomes zero

                {

                    System.out.println("DIVISION NOT POSSIBLE");

                    break;

                }

                else

                    result=a/b;

            default:

                System.out.println("YOU HAVE ENTERED A WRONG CHOICE");

        }

        System.out.println("RESULT = "+result);

    }

**9. Write a java code to copy one array into another.**

Ans:

int a[] = {1, 8, 3};

// Create an array b[] of same size as a[]

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

// Copy elements of a[] to b[]

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

b[i] = a[i];

System.out.println("Contents of a[] ");

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

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

System.out.println("\n\nContents of b[] ");

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

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

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

Ans:

**int** arr1[] = {1, 2, 3};

**int** arr2[] = {1, 2, 3, 4};

**int** len1 = arr1.length;

**int** len2 = arr2.length;

**if**(len1 > len2){

**for**(**int** i = 0; i < len1 ; i++){

System.***out***.println(arr1[i]);

}

}**else**{

**for**(**int** i = 0; i < len2 ; i++){

System.***out***.println(arr2[i]);

}

}

**11. Write a java code to display a reverse String array.**

Ans:

String input = "GeeksForGeeks";

        // convert String to character array

        // by using toCharArray

        char[] try1 = input.toCharArray();

        for (int i = try1.length-1; i>=0; i--)

            System.out.print(try1[i]);

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

Ans:

**1) Checked:** are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws*keyword.

For example, consider the following Java program that opens file at locatiobn “C:\test\a.txt” and prints first three lines of it. The program doesn’t compile, because the function main() uses FileReader() and FileReader() throws a checked exception *FileNotFoundException*. It also uses readLine() and close() methods, and these methods also throw checked exception *IOException*

|  |
| --- |
| import java.io.\*;    class Main {      public static void main(String[] args) {          FileReader file = new FileReader("C:\\test\\a.txt");          BufferedReader fileInput = new BufferedReader(file);            // Print first 3 lines of file "C:\test\a.txt"          for (int counter = 0; counter < 3; counter++)              System.out.println(fileInput.readLine());            fileInput.close();      }  } |

Output:

Exception in thread "main" java.lang.RuntimeException: Uncompilable source code -

unreported exception java.io.FileNotFoundException; must be caught or declared to be

thrown

at Main.main(Main.java:5)

To fix the above program, we either need to specify list of exceptions using throws, or we need to use try-catch block. We have used throws in the below program. Since *FileNotFoundException* is a subclass of *IOException*, we can just specify *IOException* in the throws list and make the above program compiler-error-free.

|  |
| --- |
| import java.io.\*;    class Main {      public static void main(String[] args) throws IOException {          FileReader file = new FileReader("C:\\test\\a.txt");          BufferedReader fileInput = new BufferedReader(file);            // Print first 3 lines of file "C:\test\a.txt"          for (int counter = 0; counter < 3; counter++)              System.out.println(fileInput.readLine());            fileInput.close();      }  } |

Output: First three lines of file “C:\test\a.txt”

**2) Unchecked** are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.  
In Java exceptions under *Error*and *RuntimeException*classes are unchecked exceptions, everything else under throwable is checked.

+-----------+

| Throwable |

+-----------+

/ \

/ \

+-------+ +-----------+

| Error | | Exception |

+-------+ +-----------+

/ | \ / | \

\\_\_\_\_\_\_\_\_/ \\_\_\_\_\_\_/ \

+------------------+

unchecked checked | RuntimeException |

+------------------+

/ | | \

\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

unchecked

Consider the following Java program. It compiles fine, but it throws *ArithmeticException* when run. The compiler allows it to compile, because *ArithmeticException* is an unchecked exception.

|  |
| --- |
| class Main {     public static void main(String args[]) {        int x = 0;        int y = 10;        int z = y/x;    }  } |

Output:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at Main.main(Main.java:5)

Java Result: 1

**13.  Write the difference between throw and throws with example code**

Ans:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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. |

Throw :

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either [checked or unchecked exception](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/). The throw keyword is mainly used to throw custom exceptions.

// Java program that demonstrates the use of throw

class ThrowExcep

{

    static void fun()

    {

        try

        {

            throw new NullPointerException("demo");

        }

        catch(NullPointerException e)

        {

            System.out.println("Caught inside fun().");

            throw e; // rethrowing the exception

        }

    }

    public static void main(String args[])

    {

        try

        {

            fun();

        }

        catch(NullPointerException e)

        {

            System.out.println("Caught in main.");

        }

    }

}

Throws:

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

class ThrowsExecp

{

    static void fun() throws IllegalAccessException

    {

        System.out.println("Inside fun(). ");

        throw new IllegalAccessException("demo");

    }

    public static void main(String args[])

    {

        try

        {

            fun();

        }

        catch(IllegalAccessException e)

        {

            System.out.println("caught in main.");

        }

    }

}

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

Ans:

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Syntax:

Try{ // }

Catch( excep. 1){}

Catch(excep. 2 ){}

Ex.

**try**{

**int** a[]=**new** **int**[5];

    a[5]=4;

    }**catch**(ArrayIndexOutOfBoundsException e){System.out.println(e);}

    System.out.println("other statement);

  }**catch**(Exception e){System.out.println("handeled");}

  System.out.println("normal flow..");

 }

**15.  Write a note on MultiThreading and MultiTasking**

Ans:

**MultiThreading**:

Multithreading in java is a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at same time.

2) You **can perform many operations together so it saves time**.

3) Threads are **independent** so it doesn't affect other threads if exception occur in a single thread.

## **Multitasking**

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

* Process-based Multitasking(Multiprocessing)
* Thread-based Multitasking(Multithreading)

### 1) Process-based Multitasking (Multiprocessing)

* Each process have its own address in memory i.e. each process allocates separate memory area.
* Process is heavyweight.
* Cost of communication between the process is high.
* Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

### 2) Thread-based Multitasking (Multithreading)

* Threads share the same address space.
* Thread is lightweight.
* Cost of communication between the thread is low.

**16.  Write a short note on Deque and give example code.**

Ans:

Java Deque Interface is a linear collection that supports element insertion and removal at both ends. Deque is an acronym for **"double ended queue".**

Deque<String> deque = **new** ArrayDeque<String>();

deque.add("Ravi");

deque.add("Vijay");

deque.add("Ajay");

// Traversing elements

**for** (String str : deque) {

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

}

O/P:

Ravi

Vijay

Ajay

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

Ans:

The idea is to allow type (Integer, String, … etc and user defined types) to be a parameter to methods, classes and interfaces. For example, classes like HashSet, ArrayList, HashMap, etc use generics very well. We can use them for any type.

**Generic Class:**

// 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>("GeeksForGeeks");

        System.out.println(sObj.getObject());

    }

}

O/P:

15

GeeksForGeeks

**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("GeeksForGeeks");

        // Calling generic method with double argument

        genericDisplay(1.0);

    }

}

O/P:

java.lang.Integer = 11

java.lang.String = GeeksForGeeks

java.lang.Double = 1.0

**Advantages of Generics:**

1. Code Reuse: We can write a method/class/interface once and use for any type we want.

.

1. Type Safety : Generics make errors to appear compile time than at run time (It’s always better to know problems in your code at compile time rather than making your code fail at run time).

19. Write a short note on Map Interface.

Ans:

A map contains values on the basis of key i.e. key and value pair. Each key and value pair is known as an entry. Map contains only unique keys.

Map is useful if you have to search, update or delete elements on the basis of key.

**Why** and **When**Use Maps:  
Maps are perfectly for key-value association mapping such as dictionaries. Use Maps when you want to retrieve and update elements by keys, or perform lookups by keys. Some examples:

* A map of error codes and their descriptions.
* A map of zip codes and cities.
* A map of managers and employees. Each manager (key) is associated with a list of employees (value) he manages.
* A map of classes and students. Each class (key) is associated with a list of students (value).

**20.  Write the difference between LinkedList and ArrayList.**

Ans: ArrayList and LinkedList both implements List interface and **maintains insertion order**. Both are **non synchronized** classes.

|  |  |
| --- | --- |
| **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.**

Ans:

ArrayList supports **dynamic arrays** that can grow as needed. Standard **Java arrays** are of a fixed length. After **arrays** are created, they cannot grow or shrink, which means that you must know in advance how many elements an **array** will hold.

22. What is the purpose of the System class?

Ans: The purpose of the System class is to provide access to system resources. It contains accessibility to standard input, standart output, error output streams, current time in millis, terminating the application, etc.

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.  
  
5. The System class contains several useful class fields and methods.  
  
6. **Some System class features are:**  
- Standard input  
- Standard output  
- Error output streams  
- Access to externally defined "properties"  
- A means of loading files and libraries  
- A utility method for quickly copying a portion of an array  
  
7. The System class cannot be instantiated.

**22. Which is the abstract parent class of FileWriter ?**

Ans: OutputStreamWriter

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

Ans: FileReader

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

Ans: FIleInputStream

**25. What are the differences between FileInputStream/FileOutputStream and RandomAccessFile**

Ans:

The **FileInputStream** is a byte input stream class that provides methods for reading bytes from a file. We can create an instance of this class by supplying a File or a path name, using these two constructors:

* FileInputStream(File file)
* FileInputStream(String name)

The **FileOutputStream** is a byte output stream class that provides methods for writing bytes to a file. We can create an instance of this class by supplying a File or a path name, and/or specify to overwrite or append to an existing file, using the following constructors:

* FileOutputStream(File file)
* FileOutputStream(File file, boolean append): if append is true, then the bytes will be written to the end of an existing file rather than the beginning.
* FileOutputStream(String name)
* FileOutputStream(String name, boolean append)

Java **RandomAccessFile** provides facility to both read and write data to a file. RandomAccessFile works with file as large [array of bytes](https://www.journaldev.com/770/string-byte-array-java) stored in the file system and a cursor using which we can move the file pointer position.

import java.io.IOException;

import java.io.RandomAccessFile;

public class RandomAccessFileExample {

public static void main(String[] args) {

try {

// file content is "ABCDEFGH"

String filePath = "/Users/pankaj/Downloads/source.txt";

System.out.println(new String(readCharsFromFile(filePath, 1, 5)));

writeData(filePath, "Data", 5);

//now file content is "ABCDEData"

appendData(filePath, "pankaj");

//now file content is "ABCDEDatapankaj"

} catch (IOException e) {

e.printStackTrace();

}

}

private static void appendData(String filePath, String data) throws IOException {

RandomAccessFile raFile = new RandomAccessFile(filePath, "rw");

raFile.seek(raFile.length());

System.out.println("current pointer = "+raFile.getFilePointer());

raFile.write(data.getBytes());

raFile.close();

}

private static void writeData(String filePath, String data, int seek) throws IOException {

RandomAccessFile file = new RandomAccessFile(filePath, "rw");

file.seek(seek);

file.write(data.getBytes());

file.close();

}

private static byte[] readCharsFromFile(String filePath, int seek, int chars) throws IOException {

RandomAccessFile file = new RandomAccessFile(filePath, "r");

file.seek(seek);

byte[] bytes = new byte[chars];

file.read(bytes);

file.close();

return bytes;

}

}

**26. Write a note on Channels and Buffer with example.**

Ans:

Java NIO Channels are similar to streams with a few differences:

* You can both read and write to a Channels. Streams are typically one-way (read or write).
* Channels can be read and written asynchronously.
* Channels always read to, or write from, a Buffer.

As mentioned above, you read data from a channel into a buffer, and write data from a buffer into a channel. Here is an illustration of that:

|  |
| --- |
| Java NIO: Channels and Buffers |
| **Java NIO: Channels read data into Buffers, and Buffers write data into Channels** |

## Channel Implementations

Here are the most important Channel implementations in Java NIO:

* FileChannel
* DatagramChannel
* SocketChannel
* ServerSocketChannel

The FileChannel reads data from and to files.

The DatagramChannel can read and write data over the network via UDP.

The SocketChannel can read and write data over the network via TCP.

The ServerSocketChannel allows you to listen for incoming TCP connections, like a web server does. For each incoming connection a SocketChannel is created.

## Basic Channel Example

Here is a basic example that uses a FileChannel to read some data into a Buffer:

RandomAccessFile aFile = new RandomAccessFile("data/nio-data.txt", "rw");

FileChannel inChannel = aFile.getChannel();

ByteBuffer buf = ByteBuffer.allocate(48);

int bytesRead = inChannel.read(buf);

while (bytesRead != -1) {

System.out.println("Read " + bytesRead);

buf.flip();

while(buf.hasRemaining()){

System.out.print((char) buf.get());

}

buf.clear();

bytesRead = inChannel.read(buf);

}

aFile.close();

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

Ans:

## **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.

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

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

Let's see the example of parameterized query:

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

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

# **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.

**29. Write a note on DDL, DML, DQL, DCL with code snippets.**

Ans:

SQL statements are divided into two major categories: data definition language (DDL) and data manipulation language (DML).

**Data Definition Language (DDL)** statements are used to define the database structure or schema. Some examples:

\* CREATE - to create objects in the database  
\* ALTER - alters the structure of the database  
\* DROP - delete objects from the database  
\* TRUNCATE - remove all records from a table, including all spaces allocated   for the records are removed  
\* COMMENT - add comments to the data dictionary  
\* RENAME - rename an object

**Data Manipulation Language (DML)** statements are used for managing data within schema objects. Some examples:

\* SELECT - retrieve data from the a database  
\* INSERT - insert data into a table  
\* UPDATE - updates existing data within a table  
\* DELETE - deletes all records from a table, the space for the records remain  
\* MERGE - UPSERT operation (insert or update)  
\* CALL - call a PL/SQL or [Java](http://ecomputernotes.com/java/what-is-java/what-is-java-explain-basic-features-of-java-language) subprogram  
\* EXPLAIN PLAN - explain access path to data  
\* LOCK TABLE - control concurrency

**Data Control Language (DCL)** statements. Some examples:

\* GRANT - gives user's access privileges to database  
\* REVOKE - withdraw access privileges given with the GRANT command

**Data Query Language(DQL):**

It is used to get the required data / result through query.

**28. Write a note on HTML , CSS and Javascript.**

Ans:

**HTML** provides the basic structure of sites, which is enhanced and modified by other technologies like **CSS** and **JavaScript**.

* 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

**CSS** is used to control presentation, formatting, and layout.

* **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 used to control the behavior of different elements.

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

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class H2jdbcReadDemo {

// JDBC driver name and database URL

static final String JDBC\_DRIVER = "org.h2.Driver";

static final String DB\_URL = "jdbc:h2:~/test";

// Database credentials

static final String USER = "sa";

static final String PASS = "";

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

try {

// STEP 1: Register JDBC driver

Class.forName(JDBC\_DRIVER);

// STEP 2: Open a connection

System.out.println("Connecting to database...");

conn = DriverManager.getConnection(DB\_URL,USER,PASS);

// STEP 3: Execute a query

System.out.println("Connected database successfully...");

stmt = conn.createStatement();

String sql = "SELECT id, first, last, age FROM Registration";

ResultSet rs = stmt.executeQuery(sql);

// STEP 4: Extract data from result set

while(rs.next()) {

// Retrieve by column name

int id = rs.getInt("id");

int age = rs.getInt("age");

String first = rs.getString("first");

String last = rs.getString("last");

// Display values

System.out.print("ID: " + id);

System.out.print(", Age: " + age);

System.out.print(", First: " + first);

System.out.println(", Last: " + last);

}

// STEP 5: Clean-up environment

rs.close();

} catch(SQLException se) {

// Handle errors for JDBC

se.printStackTrace();

} catch(Exception e) {

// Handle errors for Class.forName

e.printStackTrace();

} finally {

// finally block used to close resources

try {

if(stmt!=null) stmt.close();

} catch(SQLException se2) {

} // nothing we can do

try {

if(conn!=null) conn.close();

} catch(SQLException se) {

se.printStackTrace();

} // end finally try

} // end try

System.out.println("Goodbye!");

}

}

O/P:

Connecting to a selected database...

Connected database successfully...

ID: 100, Age: 18, First: Zara, Last: Ali

ID: 101, Age: 25, First: Mahnaz, Last: Fatma

ID: 102, Age: 30, First: Zaid, Last: Khan

ID: 103, Age: 28, First: Sumit, Last: Mittal

Goodbye!

**30. Describe the different approaches of String processing.**

Ans:

**public char charAt(int index)**

This method requires an integer argument that indicates the position of the character that the method returns.This method returns the character located at the String's specified index. Remember, String indexes are zero-based—for example,

String x = "airplane";

System.out.println( x.charAt(2) ); // output is 'r'

**public String concat(String s)**

This method returns a String with the value of the String passed in to the method appended to the end of the String used to invoke the method—for example,

String x = "book";

System.out.println( x.concat(" author") ); // output is "book author"

The overloaded + and += operators perform functions similar to the concat()method—for example,

String x = "library";

System.out.println( x + " card"); // output is "library card"

String x = "United";

x += " States"

System.out.println( x ); // output is "United States"

**public boolean equalsIgnoreCase(String s)**

This method returns a boolean value (true or false) depending on whether the value of the String in the argument is the same as the value of the String used to invoke the method. This method will return true even when characters in the String objects being compared have differing cases—for example,

String x = "Exit";

System.out.println( x.equalsIgnoreCase("EXIT")); // is "true"

System.out.println( x.equalsIgnoreCase("tixe")); // is "false"

**public int length()**

This method returns the length of the String used to invoke the method—for example,

String x = "01234567";

System.out.println( x.length() ); // returns "8"

**public String replace(char old, char new)**

This method returns a String whose value is that of the String used to invoke the method, updated so that any occurrence of the char in the first argument is replaced by the char in the second argument—for example,

String x = "oxoxoxox";

System.out.println( x.replace('x', 'X') ); // output is "oXoXoXoX"

**public String substring(int begin)/ public String substring(int begin, int end)**

The substring() method is used to return a part (or substring) of the String used to invoke the method. The first argument represents the starting location (zero-based) of the substring. If the call has only one argument, the substring returned will include the characters to the end of the original String. If the call has two arguments, the substring returned will end with the character located in the nth position of the original String where n is the second argument. Unfortunately, the ending argument is not zero-based, so if the second argument is 7, the last character in the returned String will be in the original String's 7 position, which is index 6. Let's look at some examples:

String x = "0123456789"; // the value of each char is the same as its index!

System.out.println( x.substring(5) ); // output is "56789"

System.out.println( x.substring(5, 8)); // output is "567"

**public String toLowerCase()**

This method returns a String whose value is the String used to invoke the method, but with any uppercase characters converted to lowercase—for example,

String x = "A New Java Book";

System.out.println( x.toLowerCase() ); // output is "a new java book"

**public String toUpperCase()**

This method returns a String whose value is the String used to invoke the method, but with any lowercase characters converted touppercase—for example,

String x = "A New Java Book";

System.out.println( x.toUpperCase() ); // output is"A NEW JAVA BOOK"

**public String trim()**

This method returns a String whose value is the String used to invoke the method, but with any leading or trailing blank spaces removed—for example,

String x = " hi ";

System.out.println( x + "x" ); // result is" hi x"

System.out.println(x.trim() + "x"); // result is "hix"

**public char[ ] toCharArray( )**

This method will produce an array of characters from characters of String object. For example

String s = “Java”;

Char [] arrayChar = s.toCharArray(); //this will produce array of size 4

**public boolean contains(“searchString”)**

This method returns true of target String is containing search String provided in the argument. For example-

String x = “Java is programming language”;

System.out.println(x.contains(“Amit”)); // This will print false

System.out.println(x.contains(“Java”)); // This will print true

Below program demonstrate all above methods.

public class StringMethodsDemo {

public static void main(String[] args) {

String targetString = "Java is fun to learn";

String s1= "JAVA";

String s2= "Java";

String s3 = " Hello Java ";

System.out.println("Char at index 2(third position): " + targetString.charAt(2));

System.out.println("After Concat: "+ targetString.concat("-Enjoy-"));

System.out.println("Checking equals ignoring case: " +s2.equalsIgnoreCase(s1));

System.out.println("Checking equals with case: " +s2.equals(s1));

System.out.println("Checking Length: "+ targetString.length());

System.out.println("Replace function: "+ targetString.replace("fun", "easy"));

System.out.println("SubString of targetString: "+ targetString.substring(8));

System.out.println("SubString of targetString: "+ targetString.substring(8, 12));

System.out.println("Converting to lower case: "+ targetString.toLowerCase());

System.out.println("Converting to upper case: "+ targetString.toUpperCase());

System.out.println("Triming string: " + s3.trim());

System.out.println("searching s1 in targetString: " + targetString.contains(s1));

System.out.println("searching s2 in targetString: " + targetString.contains(s2));

char [] charArray = s2.toCharArray();

System.out.println("Size of char array: " + charArray.length);

System.out.println("Printing last element of array: " + charArray[3]);

}

}

Output:

