Java - Full Stack

Date :
Page No.:

Section - A

1. Explain the difference b/w Byte Stream & Character Stream in java & give Example for each.

Ans

Aspect	Byte Stream	Character Stream
* Data type	8-bit raw data (binary)	16-bit Unicode characters
* Classes	InputStream / OutputStream	Reader / Writer
* Suitable for	Video, Images, Audio binary files	Text files, Unicode data
* Conversion	No conversions, works with raw bytes	Converts bytes to character automatically
* Examples	FileInputStream, FileOutputStream	FileReader, FileWriter.

2. Compare ArrayList & LinkedList, When would you use one over the other?

Ans When to Use :-

(i) Use of ArrayList when primary need is fast access to Elements by index & you perform more read operations than insertions or deletions.

(ii) Use of LinkedList when you will be performing

Date :
Page No.:

frequent insertion or deletions especially in middle of list.

Comparison of Arraylist & Linkedlist

Array list	Linked list
(1) This class uses a dynamic array to store elements	(1) This class uses a doubly linked list to store elements.
(2) Inefficient memory utilization	(2) Good memory utilization
(3) It can be One, two or multi-dimensional	(3) It can either single, double or circular
(4) Insertion operation is slow	(4) Insertion operation is fast.
(5) Less memory is used	(5) More memory is used
(6) This class implements a list interface, so it acts as a list.	(6) This class implements both the list & deque, so it acts as list & deque.
(7) Manipulation takes more time due to internal process	(7) Manipulation in LinkedList takes lesser time as No shifting the memory bits & traversing is easy.

Date :
Page No.:

3. Explain what lambda expressions are & how they improve code ~~readability~~ readability. Give simple example.

Ans A Lambda Expression is an anonymous function — a block of code without a name that can be passed as an argument to a method or stored in a variable.

* It is used to implement the single abstract method of a functional interface.

* Lambda improve readability by removing ~~bio~~ "Boilerplate" code such as syntax required for anonymous inner classes. This makes the code more concise & expressive, allowing developers to treat functionality as data & focus on what to do rather than how to setup the object.

Example :-

```
@FunctionalInterface
interface MathOperation {
    int operate(int a, int b);
}
```

Implementing it

```
MathOperation addition = (a, b) => a + b;
int result = addition.operate(10, 5);
System.out.print(result);

// Output : 15
```

Date :
Page No.:

(4.) What are Java Generics ? why are they imp -ortant for type safety ? Also explain ~~add~~ of Annotations in java. ? With Eg.

Ans Generics are feature that allows classes, interfaces & methods to be parameterised by type. This means you can define a placeholders.
e.g <T> is replaced with type (String or integer)

Importance : Generics enforce the type safety at compile time by specifying the type of collection can hold like List<String>, the compiler only accepts the string objects, which prevents "ClassCastException" errors at runtime.

Annotations :- The form of metadata that provide information about code to the compiler or JVM at runtime.

(i) For Compiler : @Overrides checks if a method correctly overrides a superclass method. &
@SuppressWarnings (suppress specific warnings).

(ii) For processing at Runtime :- @LogExecutionTime can be read using reflection to trigger specific actions.

(iii) Frameworks :- These are like Spring use annotations e.g @Autowired & @RestController to configure applications & enables dependency injection

Date :
Page No.:

(5) Explain steps required to connect java application to database using JDBC.

Ans 1. Load & Register Driver : Load the db's specific jdbc driver class into memory
Class.forName ("com.mysql.cj.jdbc.Driver");

2. Establish Connection : Use the DriverManager.getConnection() method, providing db URL, username & password.

3. Create a Statement : Create a Statement or PreparedStatement object from connection.
e.g : (connection.PrepareStatement (sql))

4. Execute the query :- Use object of statement to run your SQL query (executeQuery() for SELECT or executeUpdate() for Insert, Update, delete).

5. Process Results :- If you executed a SELECT query, iterate through the ResultSet object return to retrieve data.

6. Close connection :- Close all resources in reverse order of creation, typically with a finally block or using a try with resources statement to prevent resources leaks.

Date :
Page No.:

Section - B

1. Write a java pgm that reads a text file & prints the number of words, characters & lines present in it.

Soln

```
import java.io.BufferedReader;
   "   java.io.FileReader;
   "   java.io.IOException;
   "   java.io.FileNotFoundException;
public class WordCount {
    public static void main (String args []) {
        int lineCount = 0;
        int WordCount = 0;
        int charCount = 0;
        String fileName = "input.txt";
try (BufferedReader reader = new BufferedReader (
        new FileReader (fileName)))
    {
    String Line;
    while ((line = reader.readLine ()) != null)
    { lineCount ++;
      charCount += line.length ();
      String [] words = line.trim().split ("\\s+");
      if (!line.trim().isEmpty ()) {
          wordCount += words.length ();
    } }
      System.out.println ("Words:" + wordCount);
      System.out.println ("Characters:" + charCount);
      System.out.println ("Lines:" + lineCount);
    }
```

```
catch (FileNotFoundException e){
System.out.println("Error file not found"+fileName);
}
catch (IOException e){
System.out.println("Error reading file" + e.getMessage());
}
}
```

(2) Writ. a java pgm to store student & their details using generic collection perform i) add students ii) sort students by marks iii) Display top 3 students

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
public class Student {
    public static void main(String args[]){
List<Student> students = new ArrayList<>();
    students.add(new student(1,"Alice",85));
    students.add(new student(2,"Bob",91));
    students.add(new Student(3,"Charlie",78));
    students.add(new Student(4,"David",92));
students.sort(Comparator.comparingInt(
            Student::marks).reversed());
System.out.println("Top 3 students");
    students.stream()
        .limit(3)
        .forEach(s ->
System.out.println(s.name() + " (" + s.marks()
                + ")"));
}}
```

(3) String filter using Lambda Expression

```
import java.util.List;
import java.util.stream.Collectors;
public class StringFilter {
    public static void main (String args[])
    {
    List <String> input = List.of ("java",
                "Lambda", "code", "api",
                "expression");

List <string> output = input.stream().
                .filter(s -> s.length()>=5
                .map(String::toUpperCase);
                .collect(Collectors.toList());
            System.out.println(output);
    }
}
```

(4) Logging with Custom Annotation :—

Sol n:- File 1 : LogExecutionTime.java

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
@Retention (RetentionPolicy.Runtime)
@Target (ElementType.METHOD)
public @interface LogExecutionTime {
}
```

File 2 :— MethodRunner.java

```
public class MethodRunner {
    @LogExecutionTime
    public void ProcessData (){
        try {
            Thread.sleep (2000);
        }
        catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        } }
    public void anotherMethod (){
        System.out.println ("This method is not
                time");
    } }
```

File 3 :— AnnotationProcessor.java

```
import java.lang.reflect.Method;
public class AnnotationProcessor {
    public static void main (String args[])
    {
        MethodRunner run = new MethodRunner ();
for (Method method : run.getClass().getDeclared
                Methods ()){
    { if (method.isAnnotationPresent (LogExecutionT
                ime.class))
        { try {
            long startTime =
                System.currentTimeMillis();
```

```
method.invoke(runner);
Long endTime = System.currentTimeMillis();
Long duration = endTime - startTime;

System.out.println("method %s executed in
    %d ms", method.getName(), duration);
    } catch (Exception e) {
        {
            e.printStackTrace();
        }
    }
}
}
```

(5) CRUD operation

(1) Create small java application to perform CRUD operations on a student table in My using JDBC.

```
import java.sql.*;
public class Jdbc {
    public static void main (String args[])
    {
    try (Connection con = DriverManager.get
            Connection ("jdbc:mysql://localhost:3306/
                testdb", "root", "root"))
    {
        // Insert
    PreparedStatement ps = con.prepareStatement
        ("insert into student values (4
```

"Alice", "101", "alice@gmail.com"));
ps.setInt(1, 1);
ps.setString(2, "Bob");
ps.setInt(3, 85);
ps.executeUpdate();
System.out.println("Inserted");

// Update

PS = con.prepareStatement("Update set marks = ?
where name = 'Alice'");
ps.setInt(1, 95);
ps.setString(2, "Alice")
ps.executeUpdate();
System.out.println("Updated");

// deleted
ps = con.preparedStatement("delete from student
where name = ?");

ps.setString(1, "Alice");
ps.executeUpdate();
System.out.println("deleted record")
}
catch (Exception e){
e.printStackTrace();
}
}