BCS452: Object Oriented Programming with Java

Indicative List of Experiments

1. **Java Compiler & Eclipse Basics**
   - Use the Java compiler (`javac`) and the Eclipse IDE to write, compile, and execute a simple Java program.
2. **Command Line Arguments**
   - Create simple Java programs that take input via command line arguments to demonstrate dynamic behavior.
3. **Understanding OOP & Java Basics**
   - Build programs focusing on object-oriented concepts such as classes, objects, encapsulation, and abstraction fundamental to Java.
4. **Inheritance and Polymorphism**
   - Develop Java programs that utilize single and multilevel inheritance, method overriding, and demonstrate polymorphism.
5. **Exception Handling & Multithreading**
   - Implement robust error-handling techniques using Java's exception handling features.
   - Create and manage threads for concurrent execution and illustrate synchronization.
6. **Java Packages**
   - Organize code via the creation and use of custom and built-in Java packages.
7. **Java I/O Package**
   - Construct programs that perform file read/write and other input-output operations using the Java I/O package.
8. **Spring Framework Application**
   - Develop an industry-relevant application using the basics of the Spring Framework for dependency injection and modularity.

9. **Testing RESTful Web Services (Spring Boot)**
    - Create and test RESTful web services using Spring Boot to demonstrate backend application development.
10. **Testing Frontend Web Application with Spring Boot**
    - Implement and test a simple frontend web application integrated with Spring Boot, focusing on full-stack development basics.

These experiments are designed to give practical experience in Java and its ecosystem, covering both fundamental and advanced industry-relevant concepts.

Object Oriented Programming with Java – Lab Manual Overview (Session 2024-2025)

This lab manual is designed for students of BCS452: Object Oriented Programming with Java at ABES Engineering College, Ghaziabad (Dr. APJ AKTU, Lucknow affiliation). It outlines the course vision, values, and a detailed set of experiments to nurture technical skills, creativity, leadership, and professional excellence in engineering and computing, with a focus on sustainability and societal impact[1].

Mission & Values

- **Quality Policy:** Focuses on professional excellence, utility to industry/society, leadership, and holistic human development.
- **Values:** Respect and ethical behavior.
- **Mission (M1):** Foster healthy environment for innovation in computing and artificial intelligence fields.
- **Development Focus:** Design problem solutions mindful of public health, safety, cultural, societal, and environmental needs.

List of Experiments

S.No	Experiment Name
1	Use Java compiler and Eclipse platform to write and execute Java programs.
2	Create simple Java programs using command line arguments.
3	Understand OOP concepts and basics of Java programming.

4	Create Java programs using inheritance and polymorphism.
5	Implement error-handling techniques with exception handling and multithreading.
6	Create Java programs with the use of Java packages.
7	Construct Java programs using Java I/O package.
8	Create industry-oriented applications using Spring Framework.
9	Test RESTful web services using Spring Boot.
10	Test frontend web applications with Spring Boot.
11	Create a simple Spring application with dependency injection.
12	Write programs to read/write files using byte streams and character streams.

Experiment Highlights

- **Hands-on Environment:** Use of Eclipse IDE and command line for practical Java programming experience.
- **OOP Fundamentals:** Emphasizes object-oriented principles like classes, inheritance, and polymorphism.
- **Advanced Java Skills:** Covers exception handling, multithreading, input/output (I/O) operations, and use of Java packages for code organization.
- **Industry Tools:** Progresses from core Java to enterprise frameworks (Spring, Spring Boot), preparing students for backend (RESTful APIs) and frontend (web application) development.
- **Real-World Readiness:** Projects and experiments evolve toward practical, industry-oriented applications, reinforcing both backend and frontend integration.

Additional Program Outcomes

- **Engineering Solutions:** Develops capacity for designing and evaluating complex software systems with an eye toward sustainability and public benefit.
- **Innovation & Leadership:** Encourages students to take on challenging problem-solving and leadership roles in the computing industry.

This lab manual ensures a comprehensive, modern approach to Java programming and application development, blending core computing skills with professional and societal awareness[1].

⁂

Experiment 1: Use Java Compiler and Eclipse Platform to Write and Execute Java Program

What is the experiment about?

Is experiment mein aapko Java program ko likhna (write karna), compile karna, aur execute karna sikhaya jayega do important tools se:

- **Java Compiler (javac)**
- **Eclipse IDE (Integrated Development Environment)**

Easy Explanation (Hinglish + English Keywords)

1. **Java compiler (javac):**
   Ye ek tool hai jo aapke likhe hue Java code (`.java` file) ko machine understandable form (`.class` file) mein badalta hai. Is process ko **compilation** kehte hain.

   Example command line:

   ```
   javac HelloWorld.java
   ```

   - Yaha `HelloWorld.java` aapka source code hai.
   - Agar code mein koi error nahi hoga, toh output file `HelloWorld.class` banegi.
2. **Java Interpreter (java):**
   Compile hone ke baad, `.class` file ko **execute/run** karne ke liye `java` command use hota hai.

   ```
   java HelloWorld
   ```

   - Yeh program ko run karega aur output screen par show karega.
3. **Eclipse IDE:**
   - Eclipse ek software hai jaha aap **graphical interface** mein apna Java code likh sakte ho.
   - Isme automatic compilation hota hai jab aap code likhte hain.

    - Aap easily **Run** button click karke program chalate ho.

Example Program: Hello World

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

- Is program ka kaam hai screen par "Hello, World!" print karna.
- Yeh simple program sabse basic hai jo Java mein start karte waqt samajhna hota hai.

Step by Step Using Command Line:

1. Open command prompt or terminal.
2. Go to folder jahan aapka file saved hai.
3. Compile: `javac HelloWorld.java`
4. Run: `java HelloWorld`
5. Output aayega: `Hello, World!`

Step by Step Using Eclipse:

1. Open Eclipse IDE.
2. Create new Java Project (File > New > Java Project).
3. Create new class inside project (Right-click src > New > Class).
4. Write above HelloWorld code.
5. Click Run ▶ (Green button).
6. Output console mein dikhega: `Hello, World!`

Tricks to Learn and Recall Easily in Exam & Viva

- **Remember flow:**
    - **Write → Compile (javac) → Run (java)**
    - This sequence is important in command line Java programs.

- **Eclipse helps:**
    - Automatically compiles when you save.
    - Just click **Run**; no need to write commands.
- **Syntax tip:**
    - `main` method ka exact signature yaad rakho:

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

    - Ye program ka starting point hota hai.
- **Common error tip:**
    - Extension `.java` file ka hona zaroori hai.
    - Class name aur file name same hona chahiye.
    - In case of errors, read the message carefully.
- **Practice:**
    - 5-10 baar Hello World program likho aur run karo dono command line aur Eclipse mein, tab yaad easy hota hai.

⁂

Experiment 2: Creating Simple Java Programs Using Command Line Arguments

What is the experiment about?

Is experiment mein aapko sikhaaya jaayega ki Java program mein **command line arguments** kaise use karte hain. Matlab program chalate waqt hum extra inputs (parameters) command prompt se directly de sakte hain, jo program ke andar use hote hain.

Easy Explanation (Hinglish + English Keywords)

1. **Command Line Arguments** kya hote hain?
   Jab aap `java` command ke baad program ka naam likhte hain, tab uske aage koi extra words (parameters) likh sakte ho. Vohi parameters **command line arguments** hote hain.
   Example:

   ```
   java HelloWorld arg1 arg2
   ```

   Yaha `arg1` aur `arg2` command line arguments hain.

2. **main(String[] args)** ka role:
   Java program ke main method mein argument `String[] args` ek array hota hai jisme ye saare command line arguments store hote hain.

   - `args` = `arg1`
   - `args[^4_1]` = `arg2`
   - Aur is tarah se aage.

3. Argument ko kaise access karen?
   Aap simply `args` array ke particular index pe jaake values use kar sakte ho.

Example Program: CommandLineArgs.java

```
public class CommandLineArgs {
    public static void main(String[] args) {
        System.out.println("Total arguments: " + args.length);

        for (int i = 0; i < args.length; i++) {
            System.out.println("Argument " + i + ": " + args[i]);
        }
    }
}
```

Explanation of Example:

- Yeh program count karta hai kitne arguments diye gaye hain aur print karta hai.
- Fir ek loop ke through sab arguments ko ek-ek karke print kar deta hai.

Step-by-step Running Using Command Line:

1. Save the file as `CommandLineArgs.java`.
2. Compile:

   ```
   javac CommandLineArgs.java
   ```

3. Run with command line arguments:

   ```
   java CommandLineArgs apple orange banana
   ```

4. Output dekho:

```
Total arguments: 3
Argument 0: apple
Argument 1: orange
Argument 2: banana
```

Using Eclipse:

- Eclipse mein apne project mein class banao.
- Code likho jaise upar diya hai.
- Run button ke side mein chhota arrow pe click karo → Run Configurations → Arguments tab.
- Wahan “Program arguments” box mein likho:

  ```
  apple orange banana
  ```

- Run karo program, output console mein same result aayega.

Tricks to Learn and Recall Easily in Exam & Viva

- **main(String[] args) dhyaan se yaad karo, kyonki wahi command line arguments ko hold karta hai.**
- Jab program run karte ho, to arguments `java` command ke baad space se separate likho.
- Arguments hamisha **String array** mein store hote hain, isliye agar number lena ho to `Integer.parseInt(args[i])` ka use karna padega.
- Example:

```
int num = Integer.parseInt(args[^4_0]);
```

- Practice karne ke liye 2-3 baar likho program aur different arguments se chalakar dekho.
- **Viva Tip:** Agar poocha jaaye "Command line arguments kya hote hain?", bolo:
  "Program chalate waqt external inputs jo command prompt se diye jaate hain, unko hum command line arguments kehte hain, jo main method ke `String[] args` mein store hote hain."
- Agar error aaye to check karo arguments diye hain ya nahi! (args.length check karna important hai).
- Ek chhota code trick :-

  ```
  System.out.println(args.length > 0 ? args[^4_0] : "No argument");
  ```

Yeh code safe hai, bina arguments ke bhi error nahi dega.

⁂

Experiment 3: Understand OOP Concepts and Basics of Java Programming

What is this experiment about?

Is experiment mein aapko **Object-Oriented Programming (OOP)** ke basic concepts samajhne hain aur unko Java programming mein kaise implement karte hain, yeh sikhna hai. Saath hi Java ke fundamentals bhi cover honge.

Easy Explanation (Hinglish + English Keywords)

1. OOP Concepts (Basic)

- **Class:** Blueprint ya template jisme variables (data) aur methods (functions) hote hain.
- **Object:** Class ka ek instance ya object hota hai. Real world ka koi thing jise hum program mein represent karte hain.
- **Encapsulation:** Data (variables) ko methods ke saath bundle karna. Access modifiers (private, public) use karke data hide karna.
- **Abstraction:** Sirf necessary details dikhana, complex implementation chhupana.

- **Inheritance:** Ek class doosri class ke properties aur methods inherit kar sakti hai.
- **Polymorphism:** Ek cheez ke multiple forms hote hain. Method overloading & overriding iska example hain.

2. Java Basics

- Program ka entry point hota hai:

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

- Variables, data types, loops, conditionals.
- Syntax basics: class declaration, methods, return types.

Example Program: Simple Class and Object (OOP Basics)

```
// Class definition
public class Student {
    // Variables (Attributes)
    String name;
    int rollNumber;

    // Method to display student info
    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Roll Number: " + rollNumber);
    }

    public static void main(String[] args) {
        // Object creation
        Student student1 = new Student();

        // Assign values
        student1.name = "Rahul";
        student1.rollNumber = 101;

        // Call method
        student1.displayInfo();
    }
```

```
}
```

Explanation:

- `Student` is a **class** with two attributes `name` and `rollNumber`.
- `displayInfo()` is a **method** to print student details.
- `student1` is an **object** of class `Student`.
- Values of attributes are set for `student1`.
- Method `displayInfo()` is called to show the values.

Step-by-step How to Run this Program (Command Line)

1. Save the code as `Student.java`.
2. Compile:

   ```
   javac Student.java
   ```

3. Run:

   ```
   java Student
   ```

4. Output:

   ```
   Name: Rahul
   Roll Number: 101
   ```

Tricks to Learn and Recall Easily in Exam & Viva

- **Remember OOP pillars:** Class, Object, Encapsulation, Abstraction, Inheritance, Polymorphism.
- Always start with real world example like `Student`, `Car`, `Employee` for clarity.
- **Class = Blueprint**, **Object = Instance**. This simple phrase helps a lot.
- Practice writing main method exactly:

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

- Explain `public`, `static`, `void`, `main`, `String[] args` one by one during viva.

- Make small Java programs using simple classes and objects repeatedly, concepts clear ho jayenge.
- Always mention **how encapsulation hides data** using `private` keyword (you can add `private` later in attributes to explain).
- During viva, if asked about OOP concepts, explain with real-life examples (e.g., Car inheritance of Vehicle class).
- For abstraction, mention how user doesn't need to know engine details but can drive the car.

⁂

Experiment 3: OOP Concepts aur Java Basics ka Simple Explanation (Beginners ke liye)

Object-Oriented Programming (OOP) ka matlab hai aise programming karna jisme hum real world ki cheezon (objects) ko programs mein use karte hain.

Basic Terms easy language mein:

- **Class:** Socho class ek blueprint ya design hoti hai, jaise ghar ka naksha.
- **Object:** Class se bana hua ek actual ghar (ya object) hota hai, jise hum use kar sakte hain.
- **Attributes (variables):** Ghar ke parts jaise kamra (room), darwaza (door), waise hi class ke andar data rakhte hain.
- **Methods (functions):** Ghar ki kaam karne wali cheezein jaise light on/off karna, waise hi class ke andar functions hote hain jo kaam karte hain.

Simple example:

Student class ka example lo:

- Student ka naam (name) aur roll number (rollNumber) hote hain.
- Hum ek student ka object banayenge jisme ye details store karenge.
- Aur us student ka info print karne ka method likhenge.

Sample Code (Bahut simple):

```
public class Student {
    String name;       // Student ka naam
    int rollNumber;    // Student ka roll number

    // Method jo student ki details print karega
    public void showDetails() {
        System.out.println("Name: " + name);
        System.out.println("Roll Number: " + rollNumber);
    }

    public static void main(String[] args) {
        Student student1 = new Student();  // Student ka object banaana

        student1.name = "Amit";             // Naam assign karna
        student1.rollNumber = 10;           // Roll number assign karna

        student1.showDetails();             // Method call karke details print karna
    }
}
```

Output:

```
Name: Amit
Roll Number: 10
```

Easy Tricks to Remember:

- **Class** = Design/pattern jisme data aur functions hotey hain.
- **Object** = Class ka real example (jaise ghar ka naksha se bana hua ghar).
- `main` method se program start hota hai.
- Object banake uski properties ko set karo, fir methods call karo.
- Simple naam and number wale examples se shuruaat karo — like Student, Car, Employee etc.

Aap is tarah chhote-chhote programs practice karte raho, OOP concepts dheere-dheere samajh mein aajayenge.

Experiment 4: Inheritance & Polymorphism in Java

What is this experiment about?

Is experiment mein aapko Java mein **Inheritance** aur **Polymorphism** use karna sikhaya jayega. Ye OOP (Object Oriented Programming) ke important concepts hain jo real-life relationships aur code reuse ko easy banate hain.

Easy Explanation (Hinglish + English Keywords)

1. Inheritance (विरासत)

- **Inheritance** ka matlab hai ek class (Child ya Subclass) doosri class (Parent ya Superclass) ki properties aur methods ko use kar sakti hai.
- Isse code reuse hota hai, bar-baar same cheezein likhne ki zarurat nahi padti.
- Superclass ke methods/variables Subclass use kar sakta hai.

Keywords: `extends`, `superclass`, `subclass`

2. Polymorphism (अनेक रूप)

- **Polymorphism** ka matlab hota hai "many forms" — yani ek function/method different ways mein kaam kar sakta hai.
- Java mein mainly **method overriding** (run time) aur **method overloading** (compile time) se achieve hota hai.
- Most common viva aur practical mein method overriding poocha jaata hai.

Keywords: `overriding`, `overloading`, `@Override`, `dynamic method dispatch`

Example (Inheritance + Polymorphism):

```
// Parent Class (Super Class)
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

// Child Class (Sub Class)
class Dog extends Animal {
    // Overriding parent class method
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}

public class TestInheritance {
    public static void main(String[] args) {
        Animal a = new Animal();   // Animal ka object
        a.sound();                 // Output: Animal makes a sound

        Dog d = new Dog();         // Dog ka object
        d.sound();                 // Output: Dog barks

        // Polymorphism: Parent reference, child object
        Animal ref;

        ref = new Dog();
        ref.sound();               // Output: Dog barks (runtime polymorphism)
    }
}
```

Output:

```
Animal makes a sound
Dog barks
Dog barks
```

Explanation (Simple):

- `Animal` class mein `sound()` method hai.
- `Dog` class ne `Animal` se `extends` kiya (inheritance), aur usi method ko override kiya (polymorphism).
- Jab `Dog` ka object bana, uske `sound()` method ne "Dog barks" print kiya.
- Jab `Animal` reference mein `Dog` object assign kiya (runtime polymorphism), tab bhi overridden method chal gaya.

Tricks to Learn and Recall for Exam & Viva

- **Inheritance:**
  - Code likhte samay `extends` keyword ka use hamesha yaad rakho.
  - Ek Parent (superclass) bana ke usse ek ya zyada Child (subclass) classes bana sakte ho.
  - Vaise socho jaise "Car" ek Parent class hai, "Maruti" aur "Honda" Child classes.
- **Polymorphism:**
  - Method name same rakhna, signature same hona chahiye. Only body badalni hai (overriding).
  - Parent class ka reference = new Child class object likhne se runtime mein child wali method chalegi.
  - Use `@Override` annotation — iss se error nahi hoga agar signature galat hua.
- **Shortcut / Viva Tip:**
  - "Inheritance means sharing data and functions, polymorphism means multiple forms of functions."
  - Java does **single inheritance** only (one parent), but multiple interfaces implement kiye ja sakte hain (bonus point).
- **Practice:**
  - Ek Parent aur ek ya do Child class banao, dono mein same-named method rakho aur output check karo kaun sa method run hota hai.
- **Common error:**
  - Method signature galat likhne se overriding nahi hoti, dikkat aati hai. Signature exactly same rakho.

Aap is concept par dusre real-world examples bhi bana sakte hain (e.g., Vehicle → Car/Bike, Employee → Manager/Worker).

Experiment 5: Exception Handling & Multithreading in Java

What is this experiment about?

Is experiment mein aapko do important Java concepts sikhaaye jaayenge:

1. **Exception Handling** — Java me errors (exceptions) ko handle karna taaki program crash na ho.
2. **Multithreading** — Ek hi program me ek saath multiple tasks run karna.

Part 1: Exception Handling (Error Handling)

Easy Explanation (Hinglish + Keywords)

- **Exception:** Jab program chalate waqt koi problem (error) hoti hai, jise hum runtime error kehte hain.
- Agar aap exception ko handle nahi karenge, toh program crash kar jaayega.
- Exception handling se aap gracefully error ko pakad ke program ko chalate rakh sakte hain.

Important Keywords:

- `try` - jis block me error aane ka chance ho usko rakho
- `catch` - error aaye to usko catch karo aur handle karo
- `finally` - ye block chahe exception aaye ya nahi, hamesha execute hota hai
- `throw` - manually exception throw karne ke liye
- `throws` - method signature me batata ki exception aa sakta hai

Simple Example (Exception Handling):

```
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            int a = 10;
            int b = 0;
            int c = a / b;  // Division by zero error!
            System.out.println("Result: " + c); // Ye line nahi chalegi
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero is not allowed.");
        } finally {
            System.out.println("This block always executes.");
        }
        System.out.println("Program continues...");
    }
}
```

Output:

```
Error: Division by zero is not allowed.
This block always executes.
Program continues...
```

Part 2: Multithreading (Concurrency)

Easy Explanation (Hinglish + Keywords)

- **Thread**: Program ke andar ek chhota process jo alag se kaam karta hai.
- **Multithreading** matlab ek saath multiple threads chalana — jaise ek saath do kaam karna.
- Isse program fast aur efficient ho sakta hai, jaise apps me background tasks chalana.

Important Classes/Interfaces:

- `Thread` class
- `Runnable` interface
- `start()`, `run()` methods

Simple Example (Multithreading using Thread Class):

```
// Thread class extend karke thread banana
class MyThread extends Thread {
    public void run() {
        for(int i=1; i<=5; i++) {
            System.out.println("Child Thread: " + i);
            try {
                Thread.sleep(500);  // 0.5 second rukna
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted");
            }
        }
    }
}

public class ThreadExample {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();   // Thread start karta hai, run method background me chalna start
hota hai

        for(int i=1; i<=5; i++) {
            System.out.println("Main Thread: " + i);
            try {
                Thread.sleep(1000);  // 1 second rukna
            } catch (InterruptedException e) {
                System.out.println("Main thread interrupted");
            }
        }
    }
}
```

Output (Sample):

```
Child Thread: 1
Main Thread: 1
Child Thread: 2
Child Thread: 3
Main Thread: 2
Child Thread: 4
Child Thread: 5
Main Thread: 3
Main Thread: 4
Main Thread: 5
```

Tricks to Learn and Recall Easily in Exam & Viva

Exception Handling:

- Yaad rakho flow:
  `try` → `catch` → `finally` (optional)

- Common exceptions:
  - `ArithmeticException` (jaise divide by zero)
  - `NullPointerException`
  - `ArrayIndexOutOfBoundsException`
- Bachao tips: Always catch specific exceptions before generic ones (`Exception`)
- `throw` aur `throws` ka difference clearly samjho (throw ek exception ko manually generate karta hai, throws method signature me batata hai).
- Viva me poocha jaayega "Exception handling ka use kyun karte hain?" → Answer: "Program crash hone se bachane ke liye."

Multithreading:

- Know how to create threads in two ways:
  a. `extends Thread` class
  b. `implements Runnable` interface
- `start()` call karna zaroori hai—directly `run()` call nahi karte.
- Thread ke beech me sleep lagana aapko timing control sikhaata hai, which is useful.
- Practical exam me simple thread class banao aur `start()` karke run difference samjhao.
- Viva me poocha jaaye: "Multithreading kya hai?" Answer simple rakhna: "Ek program me ek se zyada threads ka ek sath chalna."
- Thread life cycle ke basic states yaad karo — like New, Runnable, Running, Dead.

Experiment 7: Java I/O Package ka Use Karke Program Banana

What is this experiment about?

Is experiment mein aapko Java mein **Input/Output (I/O)** operations karna sikha jaayega using **Java I/O package**. Matlab file se data read karna aur file mein data write karna.

Java I/O Basics (Hinglish + Keywords)

- **I/O** matlab Input-Output, jisse program data ko file se le sakta hai ya file mein bhej sakta hai.

- Java mein I/O ke liye `java.io` package hota hai jo bohot saare classes provide karta hai.
- Common classes:
  - **FileReader** and **BufferedReader** — for reading text files (character input)
  - **FileWriter** and **BufferedWriter** — for writing text files (character output)
  - **FileInputStream** and **FileOutputStream** — for reading/writing bytes (binary data)
- I/O operations generally **Exception handling** ke sath hote hain kyunki file nahi mil sakti, permission error aa sakti hai etc.

Example Program: File Read and Write (Using Character Streams)

```
import java.io.*;

public class FileReadWriteExample {
    public static void main(String[] args) {
        try {
            // Writing to a file
            FileWriter fw = new FileWriter("example.txt");
            BufferedWriter bw = new BufferedWriter(fw);

            bw.write("Hello, this is a sample text file.");
            bw.newLine();  // new line
            bw.write("Welcome to Java I/O programming!");
            bw.close();  // close writer to save file

            System.out.println("File written successfully.");

            // Reading from a file
            FileReader fr = new FileReader("example.txt");
            BufferedReader br = new BufferedReader(fr);

            String line;
            System.out.println("\nFile content:");
            while ((line = br.readLine()) != null) {  // read line by line
                System.out.println(line);
            }
            br.close();  // close reader
        }
        catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

Explanation (Simple):

- **FileWriter** aur **BufferedWriter** ka use karke hum `example.txt` file mein text likhte hain.
- `bw.write()` se hum lines likhte hain. `bw.newLine()` se new line milti hai.
- Close karna important hai taaki data save ho jaaye.
- File ko read karne ke liye **FileReader** aur **BufferedReader** ka use hota hai.
- `br.readLine()` se ek line ek baar mein padte hain, jab tak file ki lines khatam na ho jaye.
- Exception ke liye `try-catch` block lagana zaroori hai.

Tips and Tricks to Learn and Recall Easily in Practical Exam and Viva

- **Remember classes:**
    - Reading: `FileReader`, `BufferedReader`
    - Writing: `FileWriter`, `BufferedWriter`
    - For byte streams (images, audio etc.), use `FileInputStream`, `FileOutputStream`.
- **Close streams:** Always close your streams (`close()`) to avoid data loss or memory leaks.
- **Exception handling:** File operations ke saath `IOException` handle karna padta hai.
- **Buffered streams faster hote hain:** Kyunki ye data ko buffer mein rakhta hai before writing/reading, isse performance improve hoti hai.
- **Practice commands:**
    - `readLine()` returns a whole line as String.
    - Writing ke liye `write()` aur line break ke liye `newLine()` ya `\n` ka use.
- **Practical exam tips:**
    - File create karna, write karna, read karna, aur output dikhaana aap se poocha ja sakta hai.

    - Pehle write karo, phir read karo, output show karo.
- **Viva tip:**
    - Agar poochha, "Java I/O kya hota hai?" → "Java I/O data ko input ya output files se handle karna hota hai."
    - `"BufferedReader"` aur `"BufferedWriter"` ke baare mein bhi batao ki ye data ko fastly process karne ke liye hota hai.

Bonus: Simple byte stream example to Write and Read (Optional)

```
import java.io.*;

public class ByteStreamExample {
    public static void main(String[] args) {
        try {
            FileOutputStream fos = new FileOutputStream("data.dat");
            String data = "Byte stream data";
            byte[] b = data.getBytes();
            fos.write(b);
            fos.close();

            FileInputStream fis = new FileInputStream("data.dat");
            int i;
            System.out.print("File Data: ");
            while ((i = fis.read()) != -1) {
                System.out.print((char) i);
            }
            fis.close();
        }
        catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Experiment 8: Industry-Oriented Application using Spring Framework

What is this experiment about?

Is experiment mein aapko **Spring Framework** ka use karke ek basic industry-oriented Java application banana sikhaya jayega. Spring is one of the popular **Java Enterprise Frameworks** jo software development ko fast, modular, aur maintainable banata hai.

Easy Explanation (Hinglish + English Keywords)

1. **Spring Framework** kya hai?
    - Java ke liye ek lightweight framework.
    - Ye aapko **Dependency Injection (DI)** aur **Inversion of Control (IoC)** provide karta hai, matlab aapke objects automatically manage hote hain.
    - Modular hain: Aap alag-alag modules jese **Spring Core**, **Spring MVC**, **Spring Data**, use kar sakte ho.
    - Spring application easy to test aur maintain hoti hain, industry mein widely use hoti hai.
2. **Industry-oriented application** matlab aise applications jo **real-world business problems** solve karte hain, jaise User Management, Banking, Inventory systems, etc.
3. Spring mein basic components:
    - **Beans:** Jo classes ko Spring container manage karta hai.
    - **ApplicationContext:** Spring container jahan beans create aur manage hote hain.
    - **Annotations:** Jaise `@Component`, `@Autowired`, `@Service`, `@Repository` jo Spring ko batate hain ki beans kaise aur kahan use karna hai.

Simple Example: Spring Core Basic Application

Scenario:

User class banayenge aur uska object Spring container se get karenge, with dependency injection.

```
// User.java
package com.example;

public class User {
    private String name;

```

```java
    public User() {
        this.name = "Default User";
    }

    public void display() {
        System.out.println("User Name: " + name);
    }

    // Setter to use with DI
    public void setName(String name) {
        this.name = name;
    }
}
```

XML configuration (beans.xml) for Spring Container:

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- Bean definition -->
    <bean id="userBean" class="com.example.User">
        <property name="name" value="Rahul Kumar"/>
    </bean>
</beans>
```

Main Class to Load Spring Context and get Bean:

```java
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.example.User;

public class SpringApp {
    public static void main(String[] args) {
        // Load the Spring XML configuration file
        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");

        // Get the bean by id
        User user = (User) context.getBean("userBean");

        // Call method on the bean
        user.display();   // Output: User Name: Rahul Kumar
    }
}
```

Explanation (Simple):

- `User` class ek simple POJO (Plain Old Java Object) hai jisme `name` property hai.
- `beans.xml` mein Spring ko batate hain ki kaunse class ka object manage karna hai (bean define karte hain).
- `ApplicationContext` Spring container ka role karta hai jahan se aap beans ko access karte hain.
- `getBean` se aap Spring container se object lete hain, aur program me use karte hain.

Tricks to Learn and Recall Easily in Exam & Viva

- **Spring is all about Dependency Injection (DI):**
  Matlab objects khud create nahi karte, Spring container create karta hai aur inject karta hai.
- Remember basic Spring components: Beans, ApplicationContext, Annotations/XML configuration.
- **Bean declaration:** Yaad rakho 2 tarike — XML configuration (`beans.xml`) ya annotations (`@Component`, etc.).
- **Always explain:** "Spring helps loosely coupled code likhne mein help karta hai."
- In viva, agar poochha jaaye "Spring Framework kya hai?" bol sakte hain:
  "**A lightweight Java framework for enterprise applications jo dependency injection ko implement karta hai aur applications ko modular banata hai.**"
- Practice ek simple POJO class aur uska Spring bean banakar `getBean()` use karke output print karna.
- Start with core concepts before Spring MVC or Boot — kyunki basic understanding zaroori hai.

Bonus: Simple Spring Annotation Example (Without XML)

```
import org.springframework.stereotype.Component;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
```

```
@Component
public class User {
    private String name = "Annotation User";

    public void display() {
        System.out.println("User Name: " + name);
    }
}

@Configuration
@ComponentScan(basePackages = "com.example")
public class AppConfig {
}

public class SpringAnnotationApp {
    public static void main(String[] args) {
        ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);

        User user = context.getBean(User.class);
        user.display();  // Output: User Name: Annotation User
    }
}
```

Experiment 9: RESTful Web Services ko Test Karna Using Spring Boot

What is this experiment about?

Is experiment mein aapko **Spring Boot** ka use karke ek REST API banana, usko test karna, aur samajhna sikhaya jaata hai. **RESTful web service** woh hoti hai jo HTTP requests (GET, POST, PUT, DELETE) ko accept kar ke data bhejti hai (usually JSON format mein).

Simple Explanation (Hinglish + English Keywords)

- **Spring Boot:** Java framework jo REST APIs banane ko bahut easy bana deta hai, minimal configuration mein.
- **RESTful web service:** APIs jo HTTP ke rules follow karti hain, jaise browser URLs (GET for data, POST for submit, PUT for update, DELETE for remove).
- **Testing:** Dekhna ki API thik kaam kar rahi ya nahi — can use tools like **Postman**, browser, or Spring inbuilt test cases.

Step-by-Step Example: Create & Test a Simple REST API in Spring Boot

1. Create a Spring Boot Project

- Use Spring Initializr (https://start.spring.io/) ya Eclipse/IntelliJ mein project banao.
- Dependencies: **Spring Web**

2. Java Code Example: Simple REST Controller

```
// Greeting.java (Model)
public class Greeting {
    private long id;
    private String content;

    // Constructors, getters, setters
    public Greeting(long id, String content) {
        this.id = id;
        this.content = content;
    }
    // ... Getters/Setters ...
}

// GreetingController.java (REST Controller)
import org.springframework.web.bind.annotation.*;

@RestController
public class GreetingController {
    @GetMapping("/greeting")
    public Greeting greeting(@RequestParam(value = "name", defaultValue = "World")
String name) {
        return new Greeting(1, "Hello, " + name + "!");
    }
}
```

Ye API `GET /greeting` URL pe Java object ko JSON mein return karti hai.

3. Run the Application

- Main class mein `public static void main(String[] args)` hota hai (Spring Boot apps autostart ho jaate hain):

  ```
  @SpringBootApplication
  public class DemoApplication {
      public static void main(String[] args) {
          SpringApplication.run(DemoApplication.class, args);
      }
  }
  ```

- App chalane ke baad, browser ya Postman mein `http://localhost:8080/greeting?name=Rahul` likho.
- Output (JSON format):

  ```
  {"id":1,"content":"Hello, Rahul!"}
  ```

4. Testing REST APIs

- **Manual Tools:**
  - **Postman**, **curl** command, ya browser ka use karke test kar sakte ho.
  - Example:

    ```
    curl http://localhost:8080/greeting?name=Sunita
    ```

- **Spring Boot Automated Test Example (JUnit + MockMvc):**

```
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@SpringBootTest
@AutoConfigureMockMvc
```

```
public class GreetingControllerTest {
    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testGreeting() throws Exception {
        mockMvc.perform(get("/greeting?name=Vivek"))
               .andExpect(status().isOk())
               .andExpect(jsonPath("$.content").value("Hello, Vivek!"));
    }
}
```

- Ye code REST API ko automate karke test karta hai, ki actual response wahi aa raha hai jo expected hai[2][3][4].

Practical & Viva Tricks

- **Definition pucho toh:** "RESTful service Matlab ek aisi API Jo HTTP protocol follow karte hue data exchange karti hai."
- **Spring Boot ka fayda:** "Har cheez auto-configure ho jaati — kam code likhna padta hai, production ready app jaldi ban jaata hai."
- **Annotation yaad rakho:** `@RestController` (ya `@Controller` + `@ResponseBody`) REST API ke liye.
- **Testing Tools:** Postman, curl, ya browser for GET; MockMvc and JUnit for automation.
- **Jargon yaad karo:** Status code 200 (OK), 404 (Not Found), 500 (Server Error) — output dekhke batao.
- **Code tips:**
    - REST endpoint ke liye `@GetMapping`, `@PostMapping`, etc. use karo.
    - Testing mein assertion likhna important hai (expected vs actual result).
    - Automated test likhne se production bugs kam ho jaate hain.

⁂

Experiment 10: Test Frontend Web Application with Spring Boot

What is this experiment about?

Is experiment mein aapko sikhaaya jaayega ki kaise **Spring Boot** ka use karke ek simple **frontend web application** create karen aur usko test karen. Matlab ek basic web page banayenge jise Spring Boot backend se connect karenge aur browser mein display karenge.

Easy Explanation (Hinglish + English Keywords)

1. **Frontend Web Application** matlab aapka website/app ka wo hissa jo user ke saamne dikhta hai — jaise HTML pages, forms, buttons, etc.
2. **Spring Boot backend** ka kaam backend logic provide karna hota hai. Ye handle karta hai user requests, data processing, and web pages serve karna.
3. Spring Boot me hum **Thymeleaf** (template engine) ya simple static HTML pages use karke frontend banate hain.
4. Is experiment mein aap:
   - Simple HTML page create karoge.
   - Spring Boot controller se us page ko serve karoge.
   - Web browser me test karoge ki frontend page sahi aa raha hai ya nahi.

Important English Keywords:

- **Spring Boot Starter Web** – backend web server banata hai
- **Thymeleaf** – template engine jo HTML file ke andar Java variables inject karta hai
- **@Controller** – Spring annotation jo batata hai ki ye class web request handle karega
- **@GetMapping / @RequestMapping** – URL mapping
- **src/main/resources/templates/** – Thymeleaf templates folder
- **src/main/resources/static/** – static files (css/js/images) folder

Simple Example: Spring Boot with Frontend (Thymeleaf)

Step 1: Project Setup

- Create Spring Boot project with **Spring Web** and **Thymeleaf** dependencies.

Step 2: Create Controller Class

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller  // This class will handle web requests
public class HomeController {

    @GetMapping("/")   // Mapping for root URL
    public String homePage(Model model) {
        model.addAttribute("message", "Welcome to Spring Boot Frontend!");
        return "home"; // Returns the template file 'home.html'
    }
}
```

Step 3: Create Thymeleaf Template

Create `home.html` inside `src/main/resources/templates/`

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Spring Boot Frontend</title>
</head>
<body>
    <h1>Home Page</h1>
    <p th:text="${message}">This is a message</p>
</body>
</html>
```

Yahan `${message}` dynamically Spring controller se aaya hai.

Step 4: Run Application & Test

- Run Spring Boot main application class (`@SpringBootApplication` annotated class).
- Open browser and visit: `http://localhost:8080/`

- Aapko page dikhega:

  ```
  Home Page
  Welcome to Spring Boot Frontend!
  ```

Explanation (Simply)

- `@Controller` class web requests ko handle karta hai.
- `Model` object ke through data frontend HTML page me bhejate hain.
- `home.html` template page me Spring variable `${message}` display kiya jata hai.
- Spring Boot embedded Tomcat server ke through automatically ye page serve hota hai.

Tricks to Learn and Recall Easily in Exam & Viva

- **Remember flow:**
  - URL request → Controller → Model data bhejna → View (HTML) render hona
- **Important annotations:**
  - `@Controller` → web class
  - `@GetMapping("/")` → URL mapping
- **Template folder location:**
  - Thymeleaf templates hamesha `src/main/resources/templates/` me honi chahiye.
- **Dynamic content:**
  - `${variableName}` syntax se HTML me backend data dikhaya jata hai.
- **Run server:**
  - Spring Boot app run karo, then browser me `localhost:8080` access karo.
- **Viva Tip:**
  - "Spring Boot with Thymeleaf allow karta hai Java variables ko HTML pages me dynamically display karne ke liye, bina alag se frontend framework use kiye."
- **Hands-on tip:**
  - Simple static HTML bhi `src/main/resources/static` me rakh sakte ho; wo bhi serve hota hai automatically.

- **Debugging:**
    - Agar page blank aaye, template ka naam ya location check karo.
- **Common error:**
    - Template engine dependency nahi hone se pages load nahi honge.

Bonus: Simple Spring Boot Main Class

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class FrontendApp {
    public static void main(String[] args) {
        SpringApplication.run(FrontendApp.class, args);
    }
}
```

Isko run karna base app start karta hai.

Summary

- Experiment 10 me aap frontend web pages Spring Boot se banate ho.
- Controller `@GetMapping` method se HTML page serve karte ho.
- Thymeleaf se data dynamically HTML me show hota hai.
- Browser me URL open karke frontend test karte hain.

Experiment 11: Create a Simple Spring Application with Dependency Injection (DI)

What is this experiment about?

Is experiment mein aapko **Spring Framework** ka sabse important concept **Dependency Injection (DI)** samajhna aur implement karna sikhaya jayega. DI se object creation aur dependencies ko Spring container manage karta hai, jisse tightly coupled code loose coupled ho jata hai. Ye industry mein bahut useful hota hai.

Easy Explanation (Hinglish + English Keywords)

- **Dependency Injection (DI)** ka matlab hai:
  Jab ek object ko apni jo dependencies (other objects ya services) chahiye, wo khud create na karna pade, balki Spring framework us object ko provide kare.
- DI se code modular, maintainable aur testable ban jata hai.
- Spring mainly 3 tarike se DI karta hai:
  a. **Constructor-based injection**
  b. **Setter-based injection**
  c. **Field injection** (annotation ke sath)
- DI ke liye Spring container beans banata hai aur unhe automatically inject karta hai jahan zarurat ho.
- Iske liye aapko us class ko Spring Bean declare karna hota hai (XML config ya annotations ke through).

Keywords to Remember:

- **Bean:** Wo object jo Spring container manage karta hai.
- **IoC Container (Inversion of Control):** Jo object creation aur wiring karta hai.
- **@Component, @Service, @Repository:** Bean ko batane ke liye annotations.
- **@Autowired:** Injection ke liye annotation.
- **ApplicationContext:** Spring container jahan se beans milte hain.

Simple Example: DI using Annotations in Spring Boot

Step 1: Create a Service Class (Dependency)

```
import org.springframework.stereotype.Service;

@Service  // Declares this class as a Spring Bean
public class MessageService {
    public String getMessage() {
        return "Hello from MessageService!";
    }
}
```

Step 2: Create a Client Class which depends on Service

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component  // Declare as Spring Bean
public class MessagePrinter {

    private MessageService service;

    // Constructor-based Dependency Injection
    @Autowired
    public MessagePrinter(MessageService service) {
        this.service = service;
    }

    public void printMessage() {
        System.out.println(service.getMessage());
    }
}
```

Step 3: Main Class to get beans from Spring Context and run

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.example")  // Specify your package name here
public class AppConfig {
    public static void main(String[] args) {
        ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);

        // Get the bean
        MessagePrinter printer = context.getBean(MessagePrinter.class);

        // Call the method
        printer.printMessage();
    }
}
```

Output:

```
Hello from MessageService!
```

Explanation (Simply):

- `MessageService` ek simple service class hai, jisme ek method `getMessage()` hai.
- `MessagePrinter` class ko `MessageService` ki dependency chahiye. Usko `@Autowired` ke zariye inject kiya gaya hai (constructor injection).
- `ApplicationContext` Spring ka IoC container hai, jo beans ko manage karta hai.
- `AppConfig` ke through Spring ko batate hain ki beans kaha hain (`@ComponentScan`).
- `context.getBean()` se hum Spring container se `MessagePrinter` ka object lete hain, jisme dependency automatically inject hoti hai.
- Fir hum `printMessage()` call karte hain jo dependency ka kaam karta hai.

Tricks to Learn and Recall Easily in Practical Exam & Viva

- **Definition yaad karo:**
  "Dependency Injection means objects ko unki dependencies khud create nahi karni padti, Spring ya container unhe provide karta hai."
- **DI benefits:**
  - Loose coupling
  - Easier testing
  - Maintainability
- **Common annotations practice karo:**

    - `@Component` — bean batane ke liye
    - `@Service` — service layer class ke liye
    - `@Autowired` — inject karne ke liye
- **Constructor Injection pe zyada focus karo**, because it's widely used in industry and recommended.
- **IoC Container yaad rakho:** Jo objects ko create aur manage karta hai.
- **Code likhte waqt:**
    - `@Configuration` class likho aur `@ComponentScan` use karo
    - `ApplicationContext context = new AnnotationConfigApplicationContext()` se context banao
- **Practical exam tip:**
    - Simple service class aur client class banao, annotate karo, context me bean get karke method call karo.
- **Viva me poocha jaaye:**
    - "Spring mein DI kya hai?"
    - "DI ka fayda kya hai?"
    - "Bean kya hota hai?"
    - "Autowired ka use kahan hota hai?"

Experiment 12: Read/Write Files Using Byte Streams and Character Streams in Java

What is this experiment about?

Is experiment mein aapko **Java I/O streams** ke do types — **Byte Streams** aur **Character Streams** — ke through file handling sikhaya jaayega. Matlab kaise aap files se data **read** karte hain aur files mein data **write** karte hain.

- **Byte Streams:** File ka data ek-ek byte read/write karte hain. Ye binary data ke liye useful hai (images, audio, video, etc).

- **Character Streams:** File ke data ko characters (text) ke roop mein read/write karte hain. Ye text files ke liye use hota hai.

Important Keywords (Hinglish + English)

- **InputStream / OutputStream** — byte streams ke liye parent classes.
- **FileInputStream / FileOutputStream** — file se byte data read/write karne ke liye.
- **Reader / Writer** — character streams ke liye parent classes.
- **FileReader / FileWriter** — file se character data read/write karne ke liye.
- **BufferedReader / BufferedWriter** — buffered character streams jo fast read/write karte hain.
- **Exception Handling (IOException)** — file operations mein error handling bahut zaroori hai.

Part 1: File Write and Read Using Byte Streams (FileOutputStream & FileInputStream)

```
import java.io.*;

public class ByteStreamExample {
    public static void main(String[] args) {
        try {
            // Writing bytes to file
            FileOutputStream fos = new FileOutputStream("bytefile.dat");
            String data = "Byte stream example data.";
            byte[] bytes = data.getBytes();
            fos.write(bytes);
            fos.close();
            System.out.println("Byte data written to file.");

            // Reading bytes from file
            FileInputStream fis = new FileInputStream("bytefile.dat");
            int i;
            System.out.print("Reading byte data: ");
            while ((i = fis.read()) != -1) {   // read() returns int
                System.out.print((char) i);    // convert byte to char for display
            }
            fis.close();
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output Sample:

```
Byte data written to file.
Reading byte data: Byte stream example data.
```

Part 2: File Write and Read Using Character Streams (FileWriter & FileReader with BufferedReader/BufferedWriter)

```
import java.io.*;

public class CharacterStreamExample {
    public static void main(String[] args) {
        try {
            // Writing characters to file
            FileWriter fw = new FileWriter("charfile.txt");
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write("Character stream example text.");
            bw.newLine();
            bw.write("This is second line.");
            bw.close();
            System.out.println("Character data written to file.");

            // Reading characters from file
            FileReader fr = new FileReader("charfile.txt");
            BufferedReader br = new BufferedReader(fr);
            String line;
            System.out.println("Reading character data:");
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
            br.close();
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output Sample:

```
Character data written to file.
Reading character data:
Character stream example text.
This is second line.
```

Tricks to Learn and Recall Easily in Practical Exam and Viva

- **Byte Streams vs Character Streams:**
  - Byte Streams refer to `InputStream` and `OutputStream` classes and deal with raw binary data.
  - Character Streams refer to `Reader` and `Writer` classes and work with text data (characters).
- **Remember common classes:**
  - Byte Streams: `FileInputStream`, `FileOutputStream`
  - Character Streams: `FileReader`, `FileWriter`, `BufferedReader`, `BufferedWriter`
- **Always handle Exceptions:** `try-catch` blocks with `IOException`.
- **Use Buffered Streams for performance:** `BufferedReader` and `BufferedWriter` faster than direct `FileReader`/`FileWriter`.
- **Close your streams:** Always close streams using `.close()` — crucial for saving data and freeing resources.
- **Use read() method:**
  - Byte stream's `read()` returns an int representing a byte, returns -1 at EOF.
  - Character stream's `readLine()` reads a whole line as String.
- **Practical exam tip:**
  - Write a simple program to write some text into a file and then read the same file and print to console.
  - Use both byte and character stream versions for better understanding.

- **Viva answer example:**
    - *"Byte Streams read/write raw data one byte at a time and are used for binary files; Character Streams read/write data as characters and are mainly used for text files."*
    - *"Buffered streams improve performance by reducing number of calls to disk."*
- **Coding tip:**
    - For character streams, always prefer Buffered classes for readLine() method.
    - For byte streams, handle conversion from bytes to char carefully if you want to print text.

Common Keywords from All 12 Experiments

General Java Programming & OOP Keywords

- **Java compiler (javac)**
- **Eclipse IDE**
- **Command Line Arguments**
- **main(String[] args)**
- **Class**
- **Object**
- **Method**
- **Variable (Attributes)**
- **Encapsulation**
- **Abstraction**
- **Inheritance**
- **Polymorphism**
- **Method Overloading**
- **Method Overriding**
- **extends**
- **@Override**

- **Exception**
- **Exception Handling**
- **try**
- **catch**
- **finally**
- **throw**
- **throws**
- **Multithreading**
- **Thread**
- **Thread class**
- **Runnable interface**
- **start() method**
- **run() method**
- **sleep() method**
- **Input/Output (I/O)**
- **Java I/O Package (java.io)**
- **File**
- **FileReader**
- **FileWriter**
- **BufferedReader**
- **BufferedWriter**
- **FileInputStream**
- **FileOutputStream**
- **Byte Streams**
- **Character Streams**
- **IOException**
- **read()**

- **readLine()**
- **write()**
- **newLine()**
- **Close streams (close())**

Spring Framework & Spring Boot Keywords

- **Spring Framework**
- **Spring Boot**
- **Dependency Injection (DI)**
- **Inversion of Control (IoC)**
- **Bean**
- **ApplicationContext**
- **@Component**
- **@Service**
- **@Repository**
- **@Autowired**
- **@Configuration**
- **@ComponentScan**
- **getBean()**
- **ClassPathXmlApplicationContext**
- **AnnotationConfigApplicationContext**
- **RESTful Web Service**
- **REST API**
- **HTTP Methods: GET, POST, PUT, DELETE**
- **@RestController**
- **@Controller**
- **@GetMapping**

- **@RequestMapping**
- **@RequestParam**
- **JSON**
- **Postman**
- **JUnit**
- **MockMvc**
- **Thymeleaf**
- **Template Engine**
- **Model**
- **Static Resources (static folder)**
- **src/main/resources/templates**
- **src/main/resources/static**

Practical & Viva Specific Keywords / Concepts

- **Compilation**
- **Execution**
- **Runtime error**
- **Compile-time error**
- **Command line inputs**
- **Loose Coupling**
- **Code Reusability**
- **Code Modularity**
- **Concurrency**
- **Synchronization**
- **File reading/writing**
- **Buffered Streams (performance)**
- **Testing REST API**

- **Automated Testing**
- **Spring MVC**
- **Full-stack Development**
- **Real-world applications**
- **Industry-oriented application**

This set includes the core Java programming terms, OOP concepts, error handling & multitasking terminologies, Java I/O terms, and modern enterprise Java+Spring Framework terms — covering all experiments.

Java Programming & OOP Keywords

- **Java compiler (javac)** – Software jo Java code ko machine samajh mein aane wale code (bytecode) mein badalta hai.
- **Eclipse IDE** – Ek software jisme Java code likh sakte hain, run kar sakte hain, aur errors easily dekh sakte hain.
- **Command Line Arguments** – Inputs jo program run karte waqt space ke saath type kiye jaate hain, program ke andar use ho sakte hain.
- **main(String[] args)** – Java ka entry point (program yahi se start hota hai), yahan command line arguments bhi milte hain.
- **Class** – Ek template ya design jisme variables aur methods likhe hote hain. Jaise Car ka design.
- **Object** – Class ka real use ya instance. Jaise Car class se banni ek actual car.
- **Method** – Class ke andar likhi function-type cheez, jisse koi kaam hota hai.
- **Variable (Attributes)** – Data values jo object ya class ke andar store hoti hain. Jaise car ka color.
- **Encapsulation** – Data aur methods ko ek hi class ke andar band karna, aur data ko hide karna.
- **Abstraction** – Sirf zaroori details dikhana, baaki complexity chhupana.
- **Inheritance** – Ek class doosri class ke properties (very like features) aur methods le sakti hai. Jaise Scooter inherits Vehicle.

- **Polymorphism** – Ek method ka kai form hona; ek hi naam ke methods alag tarah se kaam karte hain.
- **Method Overloading** – Ek hi method name ho, lekin different parameters ho.
- **Method Overriding** – Parent class ka method, child class me same naam se dubara likhna.
- **extends** – Keyword jo batata hai ki koi class parent class ko inherit karti hai.
- **@Override** – Annotation jo batata hai ki method parent class ka method ko overwrite kar raha hai.
- **Exception** – Program run karte waqt aane wali error.
- **Exception Handling** – Code likhna taaki error aaye to program rukhe nahi; use manage kar sake.
- **try** – Block jahan error aane ki possibility ho, usko guard karta hai.
- **catch** – Jab try block me error aaye, to usko pakad ke handle karta hai.
- **finally** – Har hal mein (error aaye na aaye) run hone wala block.
- **throw** – Manually error signal bhejna.
- **throws** – Batata hai ki kisi method me error aa sakta hai.
- **Multithreading** – Ek hi program me ek saath kai kaam karwana.
- **Thread** – Program ke andar chalne wala ek chhotu process ya segment.
- **Thread class** – Java class jisko extend karke thread bana sakte hain.
- **Runnable interface** – Interface jisko implement karke bhi thread bana sakte hain.
- **start() method** – Thread ko actual chalane ke liye use hota hai.
- **run() method** – Thread ka kaam iss method me likha jaata hai.
- **sleep() method** – Thread ko kuch time ke liye sulaane ke liye.
- **Input/Output (I/O)** – Program me data ko andar lana ya bahar bhejna (jaise file read/write).
- **Java I/O Package (java.io)** – Java ka built-in package jisse file ya stream input/output hoti hai.
- **File** – Computer ka data store karne ka ek unit.
- **FileReader** – Character (text) file padhne ke liye class.
- **FileWriter** – Character (text) file likhne ke liye class.
- **BufferedReader** – File ya InputStream se fast reading ke liye, line-by-line padhne wala class.
- **BufferedWriter** – File ya OutputStream me fast writing ke liye.

- **FileInputStream** – File se raw bytes padhne ke liye.
- **FileOutputStream** – File me raw bytes likhne ke liye.
- **Byte Streams** – Ek-ek byte (binary data) read/write karne wale streams.
- **Character Streams** – Character (letters/text) ke liye read/write karne wale streams.
- **IOException** – Error jo file ya stream kaam karte waqt aata hai.
- **read()** – Ek character ya byte read karne ka method.
- **readLine()** – Ek poori line read karne ka method (BufferedReader me).
- **write()** – File me data likhne ka method.
- **newLine()** – Ek nayi line file me likhne ka method (BufferedWriter me).
- **Close streams (close())** – File ya stream ko band karna jab kaam khatam ho jaaye.

Spring Framework & Spring Boot

- **Spring Framework** – Java ka ek framework jo enterprise apps banane me madad karta hai, code ko simple aur maintainable rakhta hai.
- **Spring Boot** – Spring ka easy version, auto configuration ke saath, jaldi se project banane ke liye.
- **Dependency Injection (DI)** – Objects ki required cheezein (dependencies) Spring container khud provide karta hai, hum banaate nahi.
- **Inversion of Control (IoC)** – Control humare hand me nahi, Spring container me hota hai (object creation, wiring).
- **Bean** – Wo object jo Spring container manage karta hai.
- **ApplicationContext** – Spring ka environment jahan beans bante hain, manage hote hain.
- **@Component** – Batata hai ki class ek bean hai, Spring container isko manage karega.
- **@Service** – Special bean, business logic ya service ke liye (service classes ke liye).
- **@Repository** – Data access layer ke liye annotation, mostly database operations ke liye.
- **@Autowired** – Spring se automatically dependency inject karwane ke liye.
- **@Configuration** – Class ko Spring configuration ke roop me treat karne ke liye.
- **@ComponentScan** – Batata hai Spring ko ki kids package ke andar se beans scan karo.
- **getBean()** – ApplicationContext se koi bean (object) get karne ke liye.

- **ClassPathXmlApplicationContext** – XML config se ApplicationContext banane wala class.
- **AnnotationConfigApplicationContext** – Annotation based ApplicationContext (Java config class se).
- **RESTful Web Service** – Web API jo HTTP protocol ke rules follow karti hai, data ko JSON ya XML me exchange karti hai.
- **REST API** – Web app/website ka ek endpoint jisse dusre systems data le ya bhej sakte hain.
- **HTTP Methods: GET, POST, PUT, DELETE** – Web services me commonly used actions: data lana, bhejna, update karna, delete karna.
- **@RestController** – Spring class jo REST API ke web request handle karti hai.
- **@Controller** – Web controller ke liye annotation (frontend ya backend request handle karta hai).
- **@GetMapping** – GET HTTP request ke liye method mapping.
- **@RequestMapping** – URL/web request mapping ke liye annotation.
- **@RequestParam** – Request ke parameter ko method argument me lene ke liye.
- **JSON** – Simple text format jo data ko compactly represent karta hai ({"key": "value"}).
- **Postman** – Ek tool jis se REST APIs ko test kiya jaata hai.
- **JUnit** – Java ka unit testing framework.
- **MockMvc** – Spring apps ki REST APIs ko automate way se test karne ka tool.
- **Thymeleaf** – Java ka template engine; frontend HTML file me dynamic data bharne ke liye.
- **Template Engine** – Tool jo dynamic content ko HTML me bhar deta hai.
- **Model** – Data object jo backend se frontend (template) me bheja jaata hai.
- **Static Resources (static folder)** – CSS, JS, images, etc. jaise files ka folder jo directly serve hota hai.
- **src/main/resources/templates** – Thymeleaf templates ki default location.
- **src/main/resources/static** – Sare static files ki location.

Practical & Viva Specific

- **Compilation** – Java code ko machine ke liye samjhaane ke laayak banana.
- **Execution** – Program ko run karna.
- **Runtime error** – Program chalate waqt aane wali error.

- **Compile-time error** – Program banate time hi aane wali error (syntax mistakes).
- **Command line inputs** – Program run hone ke time pe diye gaye inputs.
- **Loose Coupling** – Jab classes ek dusre par kam depend karti hain (easy maintenance).
- **Code Reusability** – Pehle se likha gaya code dusri jagah use kar sakna.
- **Code Modularity** – Code ko chote, independent parts me todna.
- **Concurrency** – Ek sath kai kaam karna.
- **Synchronization** – Threads ko ek sath access se rokna, data safe rakhna.
- **File reading/writing** – File se data padhna ya usme likhna.
- **Buffered Streams (performance)** – Streams jo fast read/write karte hain by using internal buffer.
- **Testing REST API** – API sahi kaam kar rahi ya nahi, test karna.
- **Automated Testing** – Testing ko automate karna jaise code se test cases likhna.
- **Spring MVC** – Spring ka part jo web apps (frontend + backend) handle karta hai.
- **Full-stack Development** – Frontend + Backend dono banana aana.
- **Real-world applications** – Applications jo daily life ya industry ke problems solve karein.
- **Industry-oriented application** – Aisi application jo business ya company ke kaam me aaye.

BCS452: Object Oriented Programming with Java

Quick Syntax & Short Code Snippets for All 12 Experiments

Experiment 1: Java Compiler & Eclipse Basics

Syntax:

```
javac FileName.java  # Compile
java FileName        # Run
```

Short Code: Hello World

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Experiment 2: Command Line Arguments

Access command line args: `String[] args` in main method.

Short Code:

```
public class CommandLineArgs {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println("Arg " + i + ": " + args[i]);
    }
}
```

Run:

```
java CommandLineArgs arg1 arg2
```

Experiment 3: OOP Basics (Class, Object, Encapsulation)

Syntax: Class & Object creation

```
class Student {
    String name;
    int rollNo;
    void display() {
        System.out.println(name + " " + rollNo);
    }
    public static void main(String[] args) {
        Student s = new Student();
        s.name = "Amit";
        s.rollNo = 10;
        s.display();
    }
}
```

Experiment 4: Inheritance & Polymorphism

Syntax: Use `extends`, method overriding

```
class Animal {
    void sound() { System.out.println("Animal sound"); }
}

class Dog extends Animal {
    @Override
    void sound() { System.out.println("Dog barks"); }
}

public class TestInheritance {
    public static void main(String[] args) {
        Animal a = new Dog();  // Polymorphism
        a.sound();            // Output: Dog barks
    }
}
```

Experiment 5: Exception Handling & Multithreading

Exception Handling Syntax:

```
try {
    // risky code
} catch (ExceptionType e) {
    // handle error
} finally {
    // always executes
}
```

Exception Handling Example:

```
try {
    int a = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Divide by zero error");
}
```

Multithreading Syntax:

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread Running");
    }
    public static void main(String[] args) {
        new MyThread().start();
    }
}
```

Experiment 7: Java I/O Package (File Read/Write)

Write File using BufferedWriter:

```
BufferedWriter bw = new BufferedWriter(new FileWriter("file.txt"));
bw.write("Hello Java I/O");
bw.close();
```

Read File using BufferedReader:

```
BufferedReader br = new BufferedReader(new FileReader("file.txt"));
String line;
while ((line = br.readLine()) != null) {
    System.out.println(line);
}
br.close();
```

Experiment 8: Spring Framework (Basic Dependency Injection)

Bean definition (annotation-based):

```
@Component
public class User {
    public void display() { System.out.println("User bean"); }
}
```

Main class to get bean:

```
ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
User user = context.getBean(User.class);
user.display();
```

Experiment 9: Test RESTful Web Services using Spring Boot

Basic REST Controller:

```
@RestController
public class GreetingController {
    @GetMapping("/greet")
    public String greet(@RequestParam(defaultValue="World") String name) {
        return "Hello, " + name;
    }
}
```

Test URL:

```
http://localhost:8080/greet?name=Rahul
```

Experiment 10: Test Frontend Web Application with Spring Boot & Thymeleaf

Controller with Model:

```
@Controller
public class HomeController {
    @GetMapping("/")
    public String home(Model model) {
        model.addAttribute("msg", "Welcome!");
        return "home";  // home.html template
    }
}
```

Thymeleaf Template (home.html):

```
<p th:text="${msg}">Default message</p>
```

Experiment 11: Simple Spring Application with Dependency Injection (DI)

Constructor Injection Example:

```
@Component
public class MessagePrinter {
    private MessageService service;
    @Autowired
    public MessagePrinter(MessageService service) {
        this.service = service;
    }
    public void print() { System.out.println(service.getMessage()); }
}

@Service
public class MessageService {
    public String getMessage() { return "Hello DI!"; }
}
```

Experiment 12: Read/Write Files — Byte Streams and Character Streams

Write bytes:

```
FileOutputStream fos = new FileOutputStream("data.dat");
fos.write("Byte stream".getBytes());
fos.close();
```

Read bytes:

```
FileInputStream fis = new FileInputStream("data.dat");
int i;
while ((i = fis.read()) != -1) System.out.print((char) i);
fis.close();
```

Write characters:

```
BufferedWriter bw = new BufferedWriter(new FileWriter("file.txt"));
bw.write("Character stream");
bw.close();
```

Read characters:

```
BufferedReader br = new BufferedReader(new FileReader("file.txt"));
String line;
while ((line = br.readLine()) != null) System.out.println(line);
br.close();
```

General Tricks to Remember

- **Command line:** Write → Compile (`javac`) → Run (`java`) → Provide args if needed.
- **OOP:** Class = Blueprint, Object = Instance.
- Use **extends** for inheritance, **@Override** for polymorphism.
- Wrap risky code in **try-catch** blocks.
- Create threads by extending **Thread** or implementing **Runnable**, call **start()**.
- Close streams always with **.close()**.
- Spring beans: annotate with **@Component**, inject with **@Autowired**.
- REST API methods annotated with **@GetMapping**, **@PostMapping** etc.
- Thymeleaf: use `${variable}` to display backend data.
- For file operations, prefer **BufferedReader/BufferedWriter** for better performance.