Task 1: Runnable JAR

- Develop a small Java application that adds two numbers, provided by a user as inputs, and then compile it into a jar file. Package it as a runnable JAR file with a specified Main-Class in the manifest, and run the jar file.

Task 2: Library JAR

- Develop a Java application called MathLibrary, which contains functions for adding, subtracting, division and multiplication. The application should not have a main method. Compile this application into a jar file.
- Develop another separate Java application, and include the MathLibrary jar file in your application. Utilize the functions that are in the library to do the mathematical operations in the following manner:
    - System prompts the user to select the operation they want to perform.
    - System prompts the user to enter the values, and then the system does the computation and outputs the value.
- Compile the application into the following runnable Jars (do not use any dependency managers e.g. Maven).
    - Thick/Fat Jar
    - Thin Jar

Task 3: Running a JAR File with JVM Options and Command-Line Arguments

- Develop a Java application which you pass the following JVM options: Timezone, Minimum and Maximum Heap Sizes. Print the current the date and time.
- For your application, pass the following Command-Line arguments, an XML filepath for an argument -xml, and XSD filepath for argument -xsd. Validate the XML and print if it's valid or has any errors. Ensure it gracefully handles invalid or missing arguments

  e.g. java -jar my_application.jar -xml /home/dev/test.xml -xsd /home/dev/test.xsd
- Compile the application into a runnable jar.

Task 4: Classes

- Create an abstract class called Employees with the following properties:

  Employee Number, Employee Name, Net Salary

  The class should have the following abstract method: calculateSalary() and the following non-abstract method, displayEmployeeDetails() – displays Number and Name.
- Create a concrete class called FullTimeEmployee and another called PartTimeEmployee. Both should inherit Employees class and implement their own custom logic of calculating salaries and printing the output. FullTimeEmloye

- Create a fulltime and parttime employees objects and calculate their salaries.

Task 5: Interfaces

- Create an Interface called DataProcessor, which has a method process(String data)
- Create two concrete classes: JsonProcessor and XMLProcessor, which implement the interface. Have implementations of process(String data) in the classes, and read the values of each key/element in either the Json or XML provided.

Task 6: Overloading

- Create a class Geometry with overloaded methods for calculating the area of different shapes:
    - area(int side) for a square.
    - area(double radius) for a circle.
    - area(double length, double breadth) for a rectangle.
    - area(double base, double height, boolean isTriangle) for a triangle.
- Write a main() method to test each method by calling them with appropriate arguments.
- Develop a class StringFormatter with overloaded methods format() to handle:
    - format(String input) - Capitalize the first letter of each word.
    - format(String input, int repeat) - Repeat the string repeat times.
    - format(String input, String prefix, String suffix) - Add a prefix and suffix to the string.
- Use these methods to format user-provided input in multiple ways.

Task 7: Lists and ArrayLists

- Create a class called Book, with the following properties and Include appropriate constructors, getters, setters:

  Int bookId, String bookName, String bookAuthor, int numberOfCopies, Date datePublished
- Create a List which will hold Books.
- The following are a list of functionalities to achieve:

  a. Add New Book
     - New Book Entry added to the List

  b. Get Book
     - Get book at a certain position in the List

  c. Update Book Details
     - Change a property of a particular book in a particular position in the List

  d. Delete Book

- Delete a book in a particular position in the List
- Delete a book based on a certain property.

e. Display Books
   - Print all the books available in the list

f. Display Books Ordered By a certain property i.e.
   bookId, bookName, bookAuthor, numberOfCopies, datePublished

   either Ascending or Descending.

g. Display Books Filtered By a certain property i.e.
   bookId, bookName, bookAuthor, numberOfCopies, datePublished

   use streams and filters to achieve the filtering, and not looping through the list.

Task 8: Maps - I

- Create an Employee class with the following fields and include appropriate constructors, getters, setters:

  Int employeeId, String employeeNumber, String name, String department

- Create a Map, Map<String, List<Employee>>, where:
  - The key is the department name (String).
  - The value is a List<Employee> representing all employees in that department.
- The following are a list of functionalities to achieve:

  a. Add New Employee
     - Add a new Employee to the appropriate department.
     - **If the department does not exist, create a new entry in the Map**

  b. Get Employee
     - Search for an employee by id, number or name across all departments.

  c. Update Employee Details
     - Update the details of a particular employee. **The department can also be updated**.

  d. Delete Employee
     - Remove an employee by a certain property.
     - If the department becomes empty after removal, delete the department from the map.

  e. Display Employees in all departments, group by department

  f. Display Employees Ordered By a certain property either Ascending or Descending

  g. Display Employees Filtered By a certain property. Use streams and filters to achieve the filtering, and not looping through the list.

  h. Count Employees:
     - Display the total number of employees in each department.
     - Display the overall total number of employees.

Task 9: Maps – II

- Create a Column class with the following fields and include appropriate constructors, getters, setters:

  Int columnId, String columnName, String dataType

- Create a Table class with the following fields and include appropriate constructors, getters, setters:

  Int tableId, String tableName, Map<Integer, Column> columns

  For the Columns, the key is the column position.

- Create a Map, Map<String, Table> tables to hold all the Table records:
  - The key is the table name.
- **NB: Make sure the Maps you use in this exercise lexicographically order their keys i.e. maintain the order of its keys from lowest to highest**.
- The following are a list of functionalities to achieve:

  a. Add New Table and its columns
  b. Add/Edit/Remove column in a particular table.
  c. Get Tables which have similar-named columns e.g. date_created
  d. Display All Tables and their columns

Task 10: Pass By Value/Reference

- Create a Product class with the following fields and include appropriate constructors, getters, setters:

  String productName, double price

  Implement methods that illustrate **pass-by-value** for both primitives and object references:

- **Update Primitive**: Write a method void updatePrice(double price) that tries to update the price but does not affect the original variable.
- **Modify Object Reference**: Write a method void updateProduct(Product product) that changes the name and price of the passed object.
- **Reassign Object Reference**: Write a method void reassignProduct(Product product) that assigns a new object to the parameter and show that the reassignment does not affect the original object in the caller, as only the local copy of the reference is modified.
- In each of these methods, you'll first create the variable outside, pass it to the function, print the value inside, and then print outside as well, e.g.:

```
public void applyDiscount(double discount) {
    discount = discount / 2;
    System.out.println("Inside method: Discount is " + discount);
}
```

Caller:

```
double discount = 20.0;
applyDiscount(discount);
System.out.println("Outside method: Discount is " + discount);
```

Task 11: Error Handling

a. Create a program that validates user input to ensure it is a valid integer. If the input is invalid, the program should catch the exception and display an appropriate error message. If the input is valid, print the number squared. If invalid, display an error message and allow the user to try again.

b. Create a BankAccount class with the following fields and include appropriate constructors, getters, setters:

double balance

Also include the following methods:

- deposit(double amount)
- withdraw(double amount)

- Create a custom exception InsufficientFundsException
- Use try-catch to handle:
  - Invalid deposits (e.g., negative or zero amounts).
  - Withdrawals exceeding the balance, triggering InsufficientFundsException.
- Use a finally block to print a message (e.g., "Transaction complete").

c. Create an Employee class with the following fields and include appropriate constructors, getters, setters:

String employeeNumber, String employeeName, double netSalary

- Get user input to create an employee while providing the values for these properties.
- If user enters an invalid netSalary, throw the appropriate Java Exception. If user enters an employeeNumber that has a length greater than 5 characters, throw a custom EmployeeDataException. If a user enters an employeeName with less than 3 characters, throw EmployeeDataException. All these checks to be done in the same try statement, but different catch blocks for the different exceptions

Task 12: File Handling

- Create a program that reads a text file whose path is passed as a command-line argument and prints out the contents. Use appropriate error handling. Demonstrate different technics of reading from a file.
- Create a program that writes to a text file contents of a string. The file should reside in the working directory of the program.
- Create a program that has a List of Employees, List<Employee>, where the Employee class has the following properties:

  String employeeNumber, String employeeName, double netSalary

  The following are the requirements:

  - Utilize a file called employee.txt to store and read its contents, that resides in a path passed via command-line.
  - If the file exists, and has contents, read the contents and deserialize them back to the List at the start of the program, and print out the employees present in the list. If not, create the file.
  - When a new employee is created, serialize the List and write it to the file.

Task 13: Regexes - I

- Create a program that validates user input using regexes for the following properties:
  - Name – Letters and Spaces only
  - Email Address
  - Mobile Number – Must either start with 254,07,01 or +254, and with the appropriate length
- Create a program that reads one of your Java files, and prints out all the methods and properties which have private access, using regexes.
- Create a program that reads the file provided '***workflow-approval.html***', and prints out all the variables that are in []. E.g. This is a [SAMPLE] text that is a [STRING]. The program needs to print out [SAMPLE] and [STRING]. The brackets are inclusive in the output.

Task 14: Regexes – II

- Below is a sample log file containing entries in the following format:

  [2025-01-06 14:32:10] INFO User: JohnDoe, Action: Login successful
  [2025-01-06 14:33:22] ERROR User: JaneDoe, Action: Invalid password
  [2025-01-07 14:34:15] INFO User: JohnDoe, Action: Viewed profile
  [2025-01-07 14:35:15] INFO User: MaryDoe, Action: Viewed profile
  [2025-01-07 14:35:18] INFO User: JohnDoe, Action: Viewed profile

[2025-01-07 14:35:42] ERROR User: JohnDoe, Action: Unauthorized access attempt
[2025-01-07 14:36:00] INFO User: JaneDoe, Action: Logout successful
[2025-01-07 14:37:00] INFO User: JohnDoe, Action: Logout successful
[2025-01-08 14:38:00] ERROR User: JohnDoe, Action: Invalid password

- Create a program that does the following:

  Extract Specific Logs:

  - Extract all ERROR logs for 6th and 8th January 2025.
  - Use **lookaheads** to find logs where the Action contains the word password.

  Identify Repeated Actions:

  - Use **backreferences** to identify logs where a user performed the same action consecutively.

  Group and Summarize Logs:

  - Use **named capturing groups** to extract:
    - The timestamp.
    - Log level (INFO or ERROR).
    - Username.
    - Action description.
  - Display the extracted information in a readable format.

Task 15: Symmetric Encryption

- Encrypt a plain text message using AES-128-GCM.
- Decrypt the encrypted message to retrieve the original plain text.

Task 16: Asymmetric Encryption/Digital Signatures – I

- **G**enerate an RSA-2048 key pair (private and public keys).
- Encrypt a plain text message and then decrypt the encrypted text to retrieve the original plain text
- Sign a plain text message using the private key and verify if it's correct using the public key.
- Demonstrate tampering detection by modifying the message and attempting verification.

Task 17: Symmetric & Asymmetric Encryption – II

- Generate an AES key using one class, and encrypt it with a public key. Using the same AES key, encrypt a random XML string and sign it with the public key.
- In another class, decrypt the encrypted AES Key using the private key. Using this AES key, decrypt the encrypted XML string and check if it has been tampered with by using the private key.

Task 18: ProcessBuilders

a. Use a ProcessBuilder to take backups of your Exams Postgresql DB, using the pg_dump command. Please note that you are not doing database connections.

   Introduce a means to take the backups every 1 minute, and dump the backups in a folder. Once they reach 10 backups, rotate them by discarding the oldest backup, to ensure you maintain a maximum of only 10.

b. In a separate program, detect the type of operating system you application is running on. Use a processbuilder to list all running processes in the OS. Create a folder in the current directory using the mkdir command, and change directory to that folder. Using the echo command, write to a file called "running_processes.txt" all the processes running in the OS.
c. Use a processbuilder to display your disk information e.g. partitions and their sizes.

Task 19: Threads and Concurrency - I

- Create for loop to loop from 1 to 1,000,000
- For chunks of 10000 numbers, assign them to a thread, which calculates the sum of numbers in its range.
- Aggregate the results from all threads at the end to print out the grant sum from the outputs of each thread.

Task 19: Threads and Concurrency – II – Producer-Consumer

- Create a program which simulates a restaurant, whereby multiple threads represent customers placing orders (Producers), and a thread which represents the Chef taking orders and preparing them. Use a shared queue to manage the orders. Ensure thread-safe operations and inter-thread communication using synchronization.

Task 20: XML and Xpaths

XML name: signatories_model_info.xml

- Without using Xpaths:
    a. Get all elements/attributes which have field_type API_BASED. List them by their 'tag_name' attribute
    b. Count all the elements which are field_type TABLE_BASED.
    c. Get all the elements which are to be checked for duplicates and the associated fields
    d. Remove the XML elements RESTRICTED_ACCESS_NATIONALITIES_MATCH_TYPE, MAX_RESTRICTED_ACCESS_NATIONALITIES, RESTRICTED_ACCESS_NATIONALITIES
    e. Update all elements where use='MANDATORY', to be OPTIONAL
- Use Xpaths to achieve the above as well.

Task 21: XML and XSDs

XML name: pac008_sample.xml

- Create an XSD which validates the following XML.
- Use Xpaths to read the values of all the elements in the XML
- Inside element OrgnlTxRef, create another element:

```
<CdtrAcct>
       <Id>
           <Othr>
               <Id/>
           </Othr>
       </Id>
 </CdtrAcct>
```

Task 22: Maven

- Install the latest Maven and create a simple hello world project that uses Maven as a dependency manager.
- Compile the project into a runnable jar using maven commands.
- Add library Gson version 2.11 as a dependency and convert the List in Task 7 to Json and back.

Task 23: Json

Json file name: Reg-Payload.json

- Create a class which matches the attributes of the json above.
- Use Jackson json library to convert the Json above into a POJO.

Task 24: Enums

- Create an enum which represents different states of matter i.e. SOLID, LIQUID, GAS
- Create a class called Matter and have a property of the enum type. Add any other properties.
- Create an array of objects of class Matter. Loop through the array and using a switch case, create separate Lists to store the Matter objects which have the same state.

Task 25: Reflection

NB: Ensure proper package names are used

- Create an Employee class with the following fields and include appropriate constructors, getters, setters:

  Int employeeId, String employeeNumber, String name, String department
- Create a static method displayEmployeeDetails(Employee employee), in a class called Utils, and inside the function, display the employee details.
- Using reflection, create an Employee Object from class Employee and set values to it's properties by using reflection to call the setters.
- Create another object from an Employee.class *constructor* which only accepts Int employeeId, String employeeNumber i.e. Employee(Int employeeId, String employeeNumber)
- Using reflection, invoke function displayEmployeeDetails to display the details of the different objects.
- Compile the program above into a jar file
- In a separate program, dynamically load the jar file and using reflection, achieve items 3, 4 and 5

Task 26: Annotations

- Create an annotation called **Description**, whose Retention is Runtime and Target is Field. The annotation should have a property called String info. Using Task 23 above, annotate some of the fields of the class Employee and include any info describing the field, and leave others without the annotation. Using reflection, get the fields which are annotated and print out the info they contain.
- Create an annotation **ImportantMethod** to mark critical methods in a class called PaymentProcessor. Create any five methods in this class, and annotate three of them. Using reflection, get only the critical methods and execute them.
- Create an annotation called **EntityInfo**, which can annotate a class or a field. Create an **enum** called **DataType** which has the following enums: VARCHAR, INT, TEXT,BIGINT

  Create another enum called **EntityType**, which has the following enums: TABLE, COLUMN

The EntityInfo annotation needs to have the following properties:

String label, EntityType entityType, DataType dataType.

Create 3 classes that represent any database tables from the DB tasks you did and give them properties and methods. Annotate the class and the fields with the EntityInfo annotation, setting the appropriate properties for the annotation e.g. class is type TABLE and fields are type COLUMN.

Using reflection, display the classes annotated and their fields.

Task 27: Generics

- Create a generic class Item<T> that can store any type of object. Create the appropriate constructor, getters and setters. Display the item present.
- Create a generic method called printArrayContents(T[] array). Using this function, print the contents of any array passed to it.
- Create a generic interface called DataHandler<T>, and have a method called processData.

  Create two classes called StringHandler and IntegerHandler, which implement the interface. For the StringHandler, process the data by reversing it and printing it in uppercase. For IntegerHandler, process the data by reciprocating it.

Task 28: JDBC Database

- Create three programs which connect to the three different databases for Exams, Employees and Library
- In the three programs, implement functions for Insert, Update and Delete operations to manipulate data in the tables in the databases.
- Implement functions to run the selection queries as defined in the tasks:

  https://sky-dbms-learning.surge.sh/docs/projects/

Task 29: RESTful APIs

NB: Research on the following concepts:

- Resources & Endpoints
- Variables
- Parameters
- Headers (Authorization, Content-Type, Accept etc...)
- Methods (GET, POST, PUT, PATCH, DELETE, OPTIONS)
- Request/Response Bodies in the common formats (JSON/XML)

- Form-Data (Multipart)
- Status Codes
- Research on **Undertow Embedded Server**

For convention, name the endpoints in lowercase with hyphenation where needed, e.g. http://localhost:1700/test/applications/model-information/1

- Using Task 28, create JSON APIs to achieve the functionalities.

Task 30: Making HTTP Calls

- Use HTTPURLConnection class to invoke any of your APIs.
- Use Java's HttpClient to invoke any of your APIs.

Task 31: Bonus

You have been provided with a CSV file of 1.1 million records (**member_details.csv**). It has the following fields:

- ID Number
- Name
- Mobile Number
- Email Address
- Gender

The data in the CSV file is **not sanitised** and might have invalid id numbers, mobile numbers and email addresses. You are required to process the CSV file and export the data in an excel worksheet and group the data in terms of the gender values available such that each gender value has it's own excel sheet. You are also required to ensure that the excel sheets have only records with valid id numbers, mobile numbers and email addresses. You will also be required to create a sheet with the invalid id numbers, mobile numbers and email addresses.