1. Collection framework

Problem:

You are building a program to keep track of student grades. The program should allow you to add and remove students, add and remove grades for each student, and calculate the average grade for each student. You also want to be able to sort the list of students by name or by average grade. Finally, you want to be able to identify the top-performing student based on their average grade.

Solution:

To solve this problem, you could use the following data structures:

- List: To store the list of students, you could use an ArrayList<Student> to allow for efficient addition and removal of students.
- Set: To ensure that each student is only added once, you could use a HashSet<String> to ensure uniqueness by roll no.
- Map: To store the grades for each student, you could use a HashMap<String, List<Integer>> where the key is the student's name and the value is a list of their grades.

Here are the major functionalities you would need to implement:

- Adding a student: You would need to add the student to the List and roll no to the Set to ensure that they are unique.
- Removing a student: You would need to remove the student from the List and roll no from the Set, as well as remove their grades from the Map.
- Adding a grade: You would need to look up the student in the Map, add the grade to their list of grades, and update their average grade.
- Removing a grade: You would need to look up the student in the Map, remove the grade from their list of grades, and update their average grade.
- Sorting the list of students: You would need to sort the List of students by name or by average grade.
- Identifying the top-performing student: You could iterate over the Map and keep track of the student with the highest average grade.

2. Comparable & Comparator

Problem:

- Create a class, Book (title, author, yearPublished) which implements Comparable and provides a natural ordering based on the year published. Then create a list of books, sort it using the Collections.sort() method, and print the sorted list.
- Create a class, Employee (name, age, salary) and sort a list of employees based on their age first, and then salary using Comparator. Create a list of employees, sort it using the Collections.sort() method, and print the sorted list.

3. Generics

Problem:

Implement a generic stack data structure in Java.

Description:

You are required to implement a stack data structure that can store elements of any type using Java generics. The stack should be able to perform basic operations such as push, pop, and peek, and should also have a method to check if it is empty.

Requirements:

- Your implementation should be based on the Java generics concept, so that it can work with any type of data.
- You should use a generic array to store the elements in the stack.
- Your implementation should have the following methods:
  - push method to add elements to the top of the stack.
  - pop method to remove and return the element from the top of the stack.
  - peek method to return the element at the top of the stack without removing it.
  - isEmpty method to check if the stack is empty.
- You should handle stack underflow and overflow errors appropriately.

4. Serialization and Deserialization

Problem:

You have been tasked with building a Java application that will allow users to save and retrieve their personal information from a file. You need to implement the serialization and deserialization of user objects to store them in a file and retrieve them when required.

Requirements:

- Define a class called "User" with the following attributes:
  - Name (String)
  - Age (int)
  - Email (String)
  - Address (String)
- Implement the Serializable interface to allow for the serialization of the User objects.
- Implement a method called "saveUser" that takes a User object and file path as parameters and writes it to a file using serialization.
- Implement a method called "loadUser" that takes a file path as a parameter and reads the User object from the file using deserialization.
- In the main method of your application, create a new User object with some sample data and save it to a file using the "saveUser" method.

- Retrieve the User object from the file using the "loadUser" method and print out the user's information to the console.

5. File handling
Problem:

You have been tasked with creating a program that reads a file containing student grades and outputs the average grade for each student. The file contains the following information:

First column: Student ID
Second column: Student name
Third column: Grade

Write a Java program that reads the file and outputs the average grade for each student.

Instructions:

- Create a class called "StudentGradeAnalyzer".
- Implement a method called "readFile" that takes in a file name and reads the contents of the file.
- The method should parse the contents of the file and store the student grades in a data structure of your choice.
- Implement a method called "calculateAverage" that calculates the average grade for each student.
- Implement a method called "printResults" that prints the results to the console. The method should output the student ID, student name, and average grade for each student.