Java Assignment

You are tasked with conceptualizing the development of a small Java application that utilizes the concepts taught from weeks 1 to 11 of the course. This hypothetical application involves file I/O operations to manage student data, uses collections to organize data, and incorporates basic object-oriented programming concepts like classes, inheritance, and exception handling.

# Answers

Let's go through each requirement and provide a detailed explanation for it:

##  1. File I/O:

To manage a text file in Java to store student data including a student's name, ID, and marks, we use classes from the `java.io` package. Here's how the process of reading from and writing to a file in Java works:

- Writing to a file:

 - We use `File Writer` to create a file writer object, which allows us to write data to a file.

 - Then, we wrap the `File Writer` object in a `Buffered Writer` to improve performance by buffering characters for efficient writing.

 - We iterate over the `Student` objects in the `students` list and write each student's details (name, ID, marks) to the file in a specified format (e.g., CSV).

 - Finally, we close the `Buffered Writer` to ensure that any buffered data is flushed to the underlying file and the resources are released properly.

- Reading from a file:

 - We use `File Reader` to create a file reader object, which allows us to read data from a file.

 - Similar to writing, we wrap the `File Reader` object in a `Buffered Reader` for efficient reading.

 - We use a loop to read each line from the file until we reach the end (`null`).

 - Within the loop, we split each line by a delimiter (e.g., comma) to extract the student's name, ID, and marks.

 - We then print out the student's details to the console.

##  2. Collections:

For storing student data with key-value pairing (where the key is the student ID and the value is the corresponding `Student` object), a `HashMap` would be suitable. Here's why:

- HashMap:

 - Provides constant-time performance for basic operations (get, put, containsKey) on average, making it efficient for retrieving and manipulating student data.

 - Allows us to associate each student's ID with their respective `Student` object, enabling fast lookup based on the student ID.

##  3. Object-Oriented Programming:

- Student Class Design:

 - We design the `Student` class with attributes `name`, `ID`, and `marks`, encapsulating the student's details.

 - We provide getter and setter methods for each attribute to ensure proper encapsulation and data integrity.

- Inheritance and Student Record Subclass:

 - We can apply inheritance by creating a subclass `Student Record` that extends the `Student` class.

 - In the `Student Record` subclass, we can add additional functionality, such as methods to calculate average marks and determine grades, while inheriting the attributes and methods from the base `Student` class.

 - This allows for code reuse, extensibility, and better organization of code.

##  4. Exception Handling:

- Error Handling:

 - We implement error handling using try-catch blocks to handle potential I/O errors and other runtime exceptions that may occur during file operations or data processing.

 - By wrapping the file, I/O operations and other critical sections of code in try-catch blocks, we can gracefully handle exceptions and prevent the program from crashing.

 - We provide informative error messages or handle exceptions appropriately based on the specific error encountered, ensuring robust application performance.

##  Rationale:

- Choice of Collection Type:

 - We chose `HashMap` for storing student data due to its efficient key-value pairing and constant-time performance for basic operations.

- Structure of Class Hierarchy:

 - We designed the `Student` class with encapsulated attributes and methods to represent a basic student entity.

 - We applied inheritance by creating a subclass `Student Record` to extend the functionality of the base `Student` class, enabling additional features like calculating average marks and determining grades.

- Use of Try-Catch Blocks:

 - We used try-catch blocks to handle potential errors during file I/O operations and other critical sections of code, ensuring robustness and preventing unexpected program termination.

By following these design decisions, we ensure that the Java application effectively manages student data, maintains code organization, and handles errors gracefully, thereby enhancing its maintainability, scalability, and robustness.

This explanation provides a comprehensive overview of how each concept is implemented in the Java application, along with the rationale behind the design decisions made.

# Program Code

import java.io.\*;

import java.util.\*;

// Student class

class Student {

    private String name;

    private String ID;

    private int marks;

    // Constructor

    public Student(String name, String ID, int marks) {

        this.name = name;

        this.ID = ID;

        this.marks = marks;

    }

    // Getter methods

    public String getName() {

        return name;

    }

    public String getID() {

        return ID;

    }

    public int getMarks() {

        return marks;

    }

    // Setter methods

    public void setName(String name) {

        this.name = name;

    }

    public void setID(String ID) {

        this.ID = ID;

    }

    public void setMarks(int marks) {

        this.marks = marks;

    }

}

public class Main {

    public static void main(String[] args) {

        // Example usage

        ArrayList<Student> students = new ArrayList<>();

        // Adding students

        students.add(new Student("John Doe", "123", 85));

        students.add(new Student("Jane Smith", "456", 92));

        students.add(new Student("Alice Johnson", "789", 78));

        // Writing to file

        try {

            FileWriter writer = new FileWriter("students.txt");

            BufferedWriter bufferedWriter = new BufferedWriter(writer);

            for (Student student : students) {

                bufferedWriter.write(student.getName() + "," + student.getID() + "," + student.getMarks() + "\n");

            }

            bufferedWriter.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

        // Reading from file

        try {

            FileReader reader = new FileReader("students.txt");

            BufferedReader bufferedReader = new BufferedReader(reader);

            String line;

            while ((line = bufferedReader.readLine()) != null) {

                String[] data = line.split(",");

                System.out.println("Name: " + data[0] + ", ID: " + data[1] + ", Marks: " + data[2]);

            }

            bufferedReader.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}