Student Record Management System using HashMap

You are assigned to develop a Java-based **Student Record Management System** that allows administrators to manage student information using a `HashMap<Integer, String>`. Each student is uniquely identified by a **Student ID (key)**, and the corresponding value is the **student's name**.

The system supports the following operations:

- Adding new students to the system.
- Removing a student using their ID.
- Finding a student by their ID.
- Updating the name of a student using their ID.
- Displaying the entire student record.

If an invalid student ID is provided for removal, lookup, or update, the system must throw a custom exception **StudentNotFoundException** with a proper message.

Project Structure and Files

The solution must include the following files:

1. **StudentManager.java**

   Contains the core logic using a `HashMap<Integer, String>`.
2. **StudentNotFoundException.java**

   Custom exception thrown when a given Student ID is not found.
3. **StudentApp.java**

   Reads input, interacts with `StudentManager`, and handles operations and exceptions.

Class Responsibilities

StudentManager.java

Field:

`HashMap<Integer, String> studentRecords`

Methods:

```java
public void addStudent(int id, String name)
public void removeStudent(int id) throws StudentNotFoundException
public void findStudent(int id) throws StudentNotFoundException
public void updateStudentName(int id, String newName) throws StudentNotFoundException
public void displayAllStudents()
```

StudentNotFoundException.java

A custom exception class extending Exception.

Constructor:

```java
public StudentNotFoundException(String message)
```

StudentApp.java

- Reads user input and operation
- Calls appropriate methods on `StudentManager`
- Catches and handles `StudentNotFoundException`
- Always prints the final student records

Functional Requirements

Add Student

- Accept Student ID and Name and store them in the HashMap.
- Duplicate IDs should overwrite the existing name.

Remove Student

- Remove a student by their ID.
- If the ID is not present, throw `StudentNotFoundException`.

Find Student

- Retrieve and display the name using Student ID.
- If the ID is not present, throw `StudentNotFoundException`.

Update Student Name

- Update the name of an existing student using their ID.
- If the ID is not present, throw `StudentNotFoundException`.

Display Students

- Display the entire student record as a HashMap.

Input Format:

- An integer 'N' (number of students to be added).
- The next '2*N' lines contain:
    - An integer (Student ID).
    - A string (Student Name).
- A string indicating the operation ("REMOVE", "FIND", or "UPDATE").

Based on the operation:

- `"REMOVE"` → An integer (Student ID) to remove.
- `"FIND"` → An integer (Student ID) to find.
- `"UPDATE"` → An integer (Student ID) to update, followed by a string (New Name).

Note: `'REMOVE'`, `'FIND'` and `'UPDATE'` are case-sensitive.

Output Format:

For Adding Students:

`Student ID [StudentID] added with name '[Name]'.`

For Removing a Student:

- If found: `Student ID [StudentID] removed.`
- If not found: `Student ID [StudentID] not found in the system.`

For Finding a Student:

- If found: `Student ID [StudentID] - Name: [Name]`
- If not found: `Student ID [StudentID] not found in the system.`

For Updating a Student's Name:

- If found: `Student ID [StudentID] updated to name '[NewName]'.`
- If not found: `Student ID [StudentID] not found in the system.`

Final Display of Students:

`Current Students: {StudentID1=Name1, StudentID2=Name2, ...}`

Sample Input 1

```
2
101
Alice
102
Bob
REMOVE
101
```

Sample Output 1

```pgsql
Student ID 101 added with name 'Alice'.
Student ID 102 added with name 'Bob'.
Student ID 101 removed.
Current Students: {102=Bob}
```

Sample Input 2

```
3
201
John
202
Emma
203
Ryan
FIND
202
```

Sample Output 2

```pgsql
Student ID 201 added with name 'John'.
Student ID 202 added with name 'Emma'.
Student ID 203 added with name 'Ryan'.
Student ID 202 - Name: Emma
Current Students: {201=John, 202=Emma, 203=Rya
```

Sample Input 3

```sql
2
301
Olivia
302
Liam
UPDATE
302
Sophia
```

Sample Output 3

```pgsql
Student ID 301 added with name 'Olivia'.
Student ID 302 added with name 'Liam'.
Student ID 302 updated to name 'Sophia'.
Current Students: {301=Olivia, 302=Sophia}
```

Sample Input 4

```sql
2
301
Mason
302
Ava
UPDATE
303
Noah
```

Sample Output 4

```pgsql
Student ID 301 added with name 'Mason'.
Student ID 302 added with name 'Ava'.
Student ID 303 not found in the system.
Current Students: {301=Mason, 302=Ava}
```