Introduction to Databases AP Project

< AP Fall 2024 >

< Instructor: Dr.Vahidi Asl >

< Presenter: Amirreza Yazdanpanah>

Table of contents

01 What are Databases?

02 Different types of Databases

03 Flat-file Databases

01 { ..

What are Databases?

< Just in case >

} ..

What are Databases?

Databases are essential systems for storing, managing, and retrieving vast amounts of information efficiently. They enable organizations to organize data in a structured way, making it easily accessible for analysis, reporting, and decision-making.

Databases can range from small, personal systems to large, enterprise-level solutions that support millions of transactions per second. By using various database models like **relational** and **NoSQL**, businesses can handle different types of data and ensure scalability, security, and performance.

Key Points!

- Store and organize data efficiently
- Allow easy access, updates, and analysis of information
- Support different database models (relational, NoSQL, …)
- Handle everything from small to large-scale data needs
- Ensure scalability, security, and performance

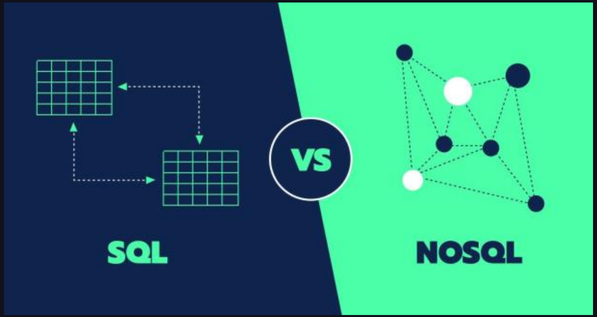
02 {..

Different types of Databases

}..

Different types of Databases

There are several types of databases, each designed to handle different types of data and serve various purposes. The two most popular are **Relational** and **NoSQL** databases.



Relational Database (RDBMS)

Structure

Data is stored in tables (rows and columns) with predefined relationships between them.

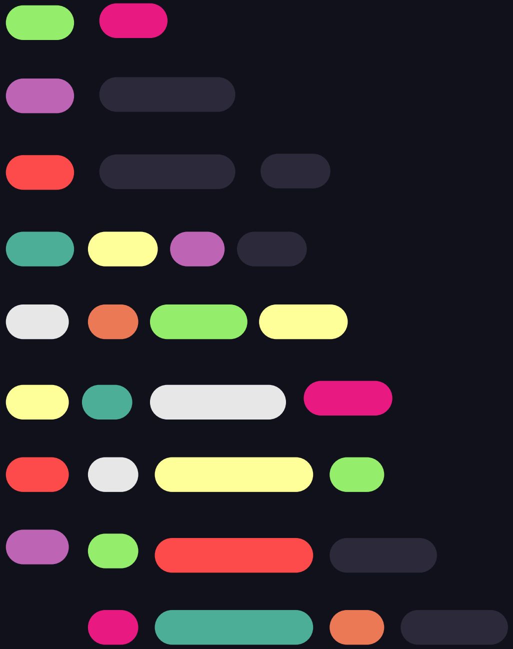
Data Model

Follows a strict schema with structured data. Every record follows the same format.

Query Language

Uses SQL (Structured Query Language) to query, insert, update, and delete data.

When to Use Relational Databases:



- When you need to handle structured data with relationships (e.g., financial systems, customer data).
- When transactional integrity and consistency are crucial.
- When the data structure is well-defined and doesn't change often.

Relational database examples

MySQL™

PostgreSQL

ORACLE

NoSQL Database

Structure

Can store unstructured, semi-structured, or structured data in various formats like documents, key-value pairs, graphs, or wide-column stores.

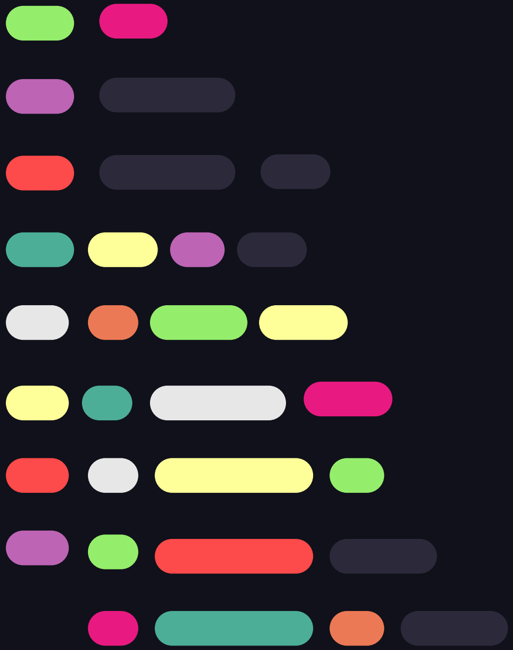
Data Model

Schema-less, allowing for flexible data models that can easily adapt to changes.

Query Language

No standard query language; query methods vary by database type (e.g., MongoDB uses its own query language, Cassandra uses CQL).

When to Use NoSQL Databases:



- When handling large volumes of unstructured or semi-structured data (e.g., social media data, IoT, logs).
- When flexible and evolving data models are needed.
- When high performance and horizontal scalability are more important than strict consistency.

NoSQL database examples

//Document

//Wide-column

//Graph

//Key-value

Key differences:

Feature	RDBMS	NoSQL
Data Structure	Tables with rows and columns	Documents, key-value pairs, graphs, or wide-column
Schema	Fixed schema	Schema-less, flexible data models
Query Language	SQL	No standard language (varies by database)
Best for	Structured, relational data	Unstructured or rapidly changing data

Relational vs NoSQL database

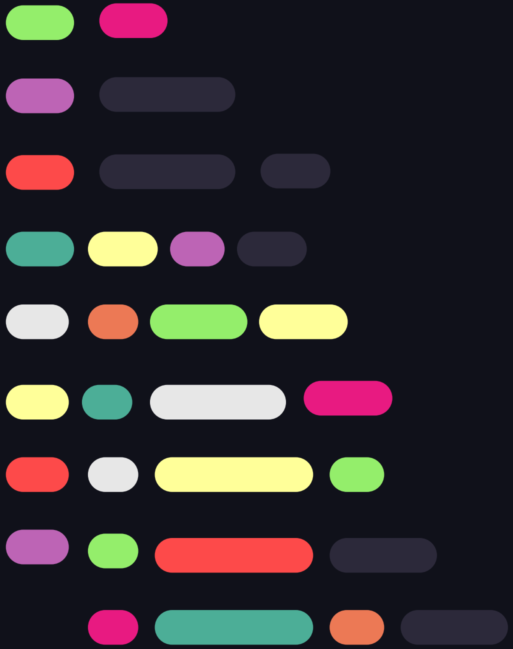
Relational databases are ideal for structured, consistent data with complex relationships, while NoSQL databases offer flexibility and scalability for unstructured or large-scale data scenarios.

03 {..

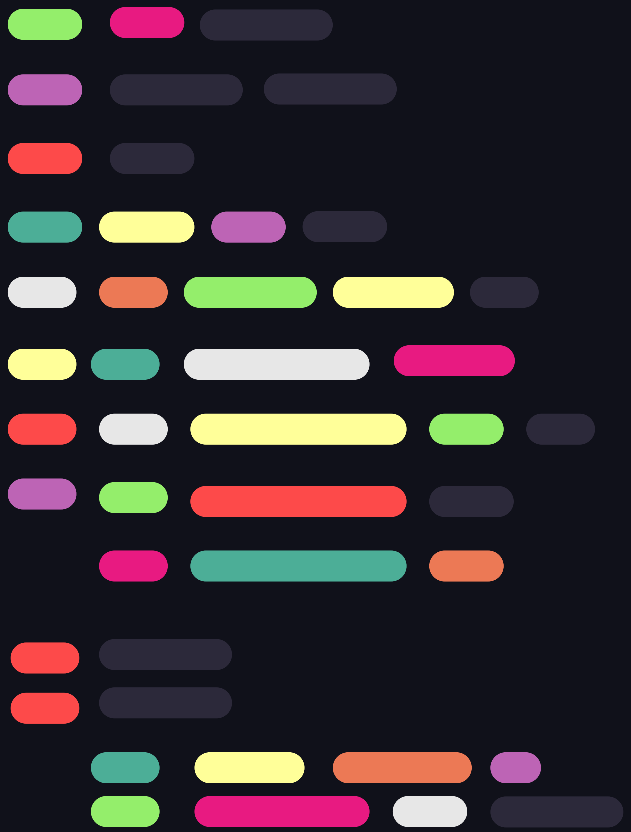
Flat-file Databases

}..

Flat-file Databases



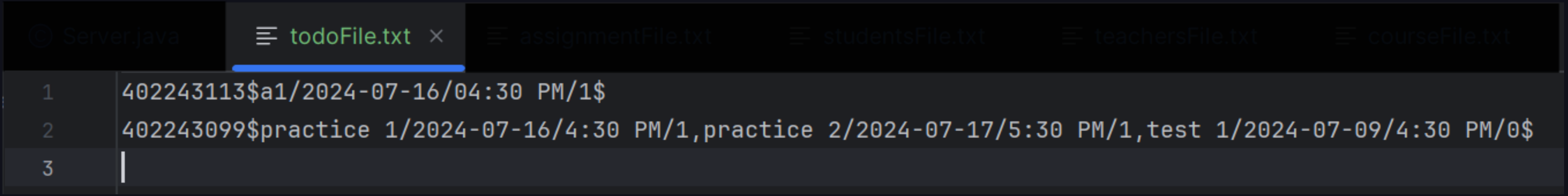
A **flat file database** is a simple type of database that stores data in a single table or plain text file, where each line holds a record, and fields are typically separated by commas, tabs, or another delimiter. Unlike more complex database systems, flat file databases do not have relationships between data sets or multiple tables.



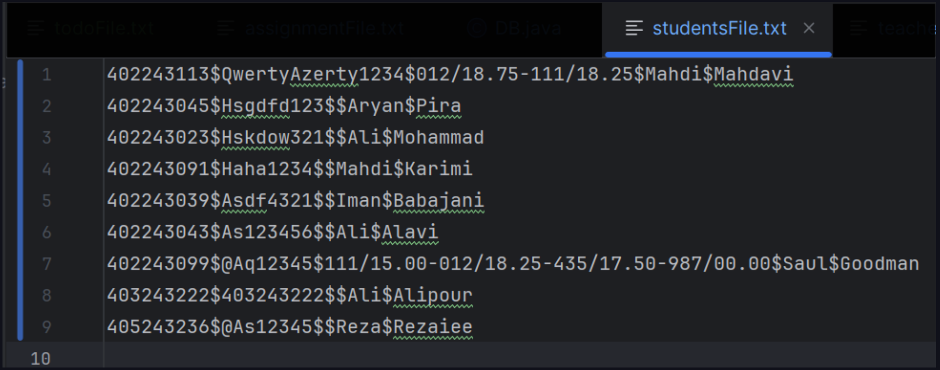
Whoa!}

< for this project, a flat-file database using .txt files is compulsary, due to its simpler nature. Implementing any of the other types of databases is bonus! >

Text File Example

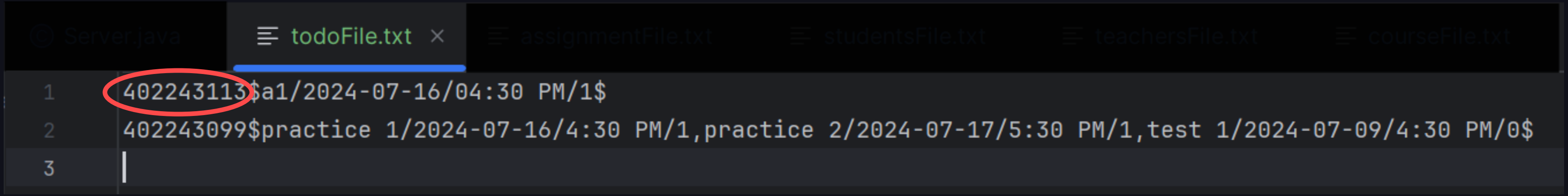


```
402243113$a1/2024-07-16/04:30 PM/1$
402243099$practice 1/2024-07-16/4:30 PM/1,practice 2/2024-07-17/5:30 PM/1,test 1/2024-07-09/4:30 PM/0$
```

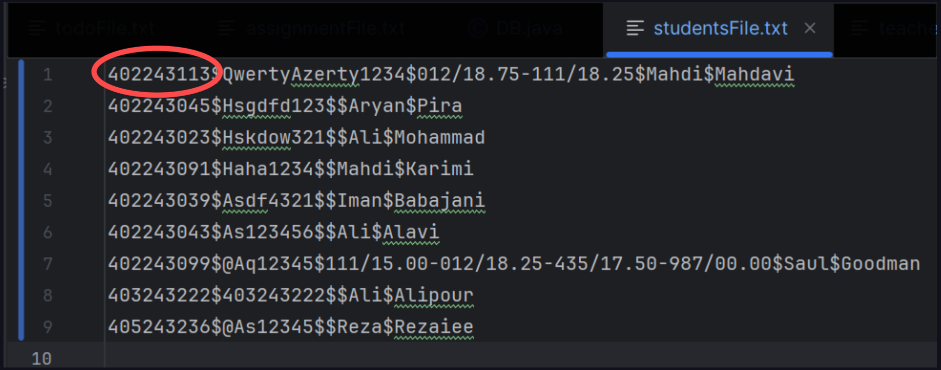


```
402243113$QwertyAzerty1234$012/18.75-111/18.25$Mahdi$Mahdavi
402243045$Hsgdfd123$$Aryan$Pira
402243023$Hskdow321$$Ali$Mohammad
402243091$Haha1234$$Mahdi$Karimi
402243039$Asdf4321$$Iman$Babajani
402243043$As123456$$Ali$Alavi
402243099$@Aq12345$111/15.00-012/18.25-435/17.50-987/00.00$Saul$Goodman
403243222$403243222$$Ali$Alipour
405243236$@As12345$$Reza$Rezaiee
```

Text File Example

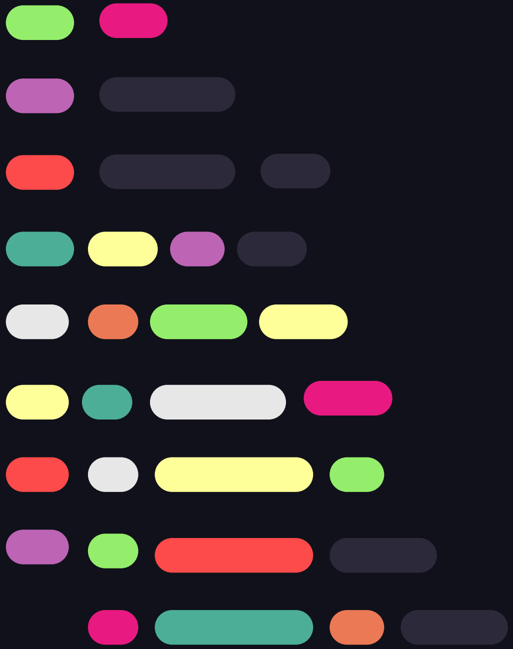


```
402243113$a1/2024-07-16/04:30 PM/1$
402243099$practice 1/2024-07-16/4:30 PM/1,practice 2/2024-07-17/5:30 PM/1,test 1/2024-07-09/4:30 PM/0$

```



```
402243113$QwertyAzerty1234$012/18.75-111/18.25$Mahdi$Mahdavi
402243045$Hsgdfd123$$Aryan$Pira
402243023$Hskdow321$$Ali$Mohammad
402243091$Haha1234$$Mahdi$Karimi
402243039$Asdf4321$$Iman$Babajani
402243043$As123456$$Ali$Alavi
402243099$@Aq12345$111/15.00-012/18.25-435/17.50-987/00.00$Saul$Goodman
403243222$403243222$$Ali$Alipour
405243236$@As12345$$Reza$Rezaiee

```

Text File Example

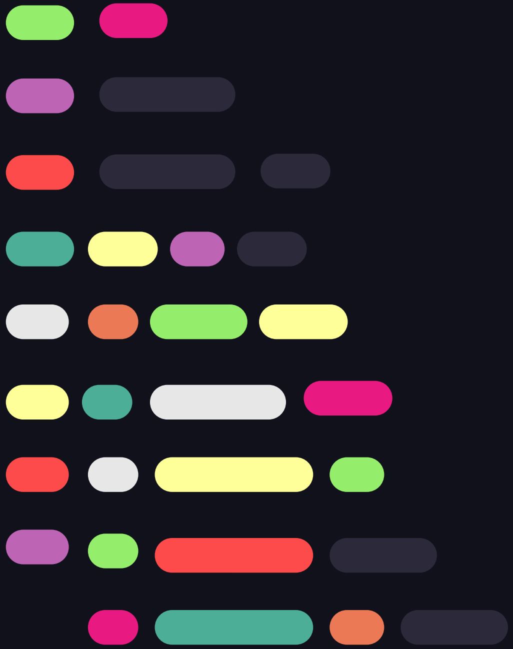


In the two text files in the previous page, both have a **Student Number** property.
You might ask why do both these files which have different purposes have the same property?
Well, this is a way to make relations between “tables” in flat-file databases. The student number property is considered the **Primary key** in the students file, and a **Foreign key** in the todo file. But what are Primary and Foreign keys?

Primary Key

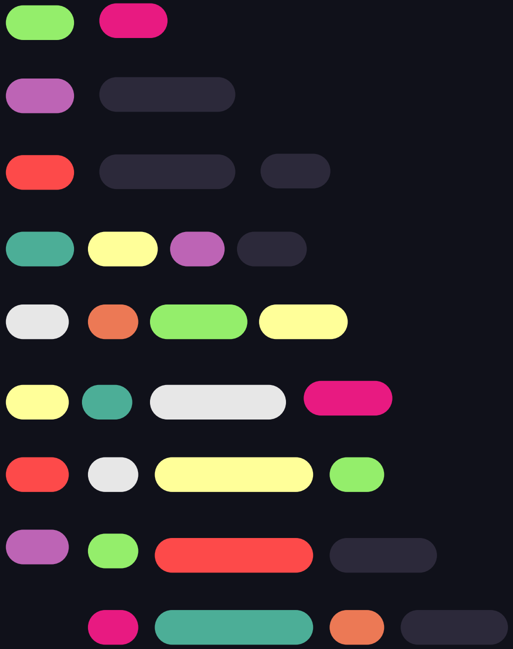
A primary key is a unique identifier for each record in a database table. It ensures that no two records in the table can have the same value for the primary key, which allows for efficient retrieval, updates, and management of data.

Foreign Key



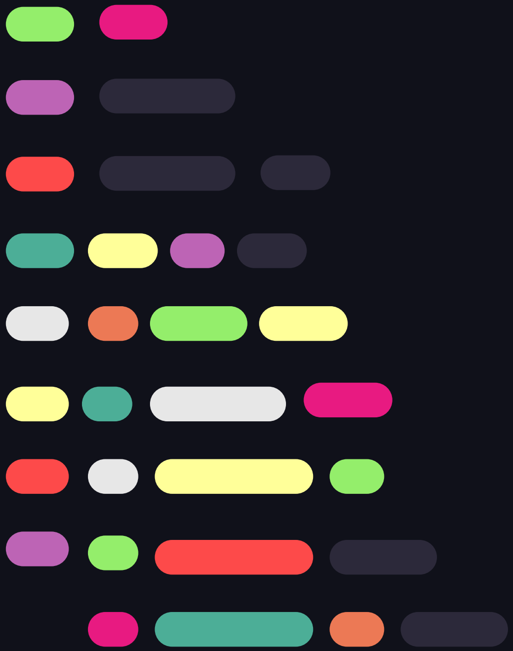
A foreign key is a field (or group of fields) in one table that establishes a relationship between that table and another table by referencing the primary key of the second table. It ensures data integrity by enforcing a link between records in the two tables, maintaining consistency in the database.

Text File Example



In this example, the student number is the unique identifier for each student, and is used to identify which student each todo task is for.

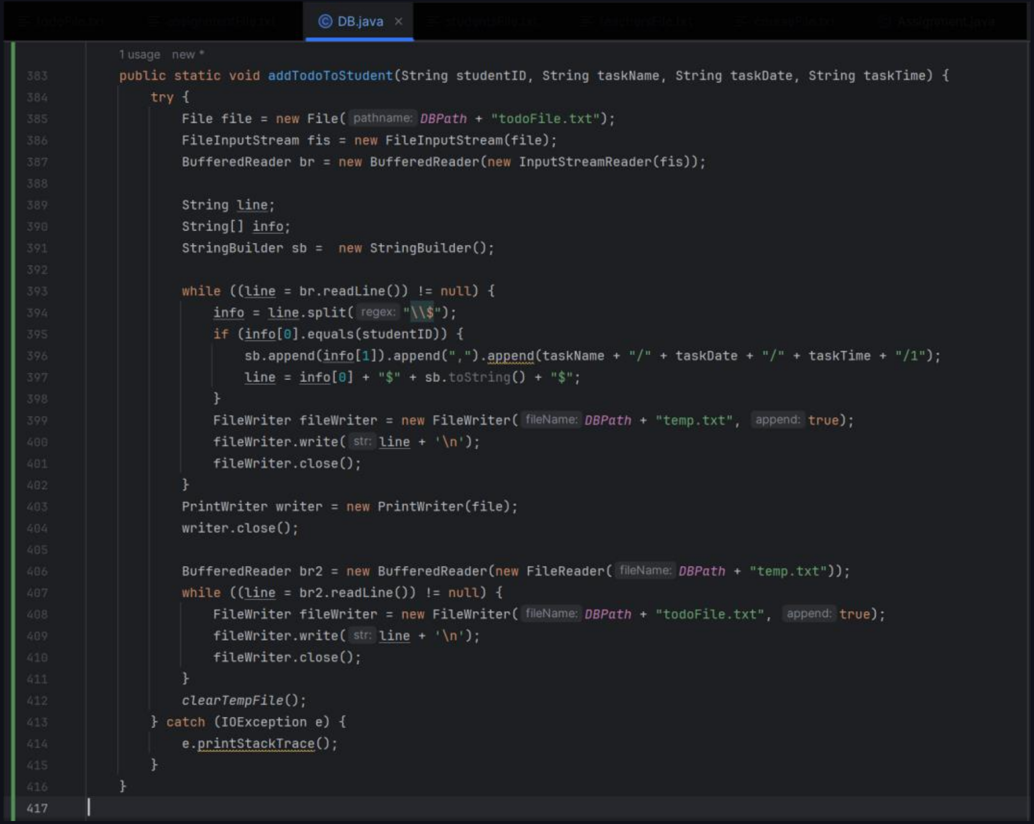
CRUD Operations



CRUD stands for Create, Read, Update, and Delete, which are the four basic operations that can be performed on data in a database or storage system. CRUD operations are fundamental to interacting with databases, whatever the database is.

In the following pages you will see implementations of CRUD for a flat-file database.

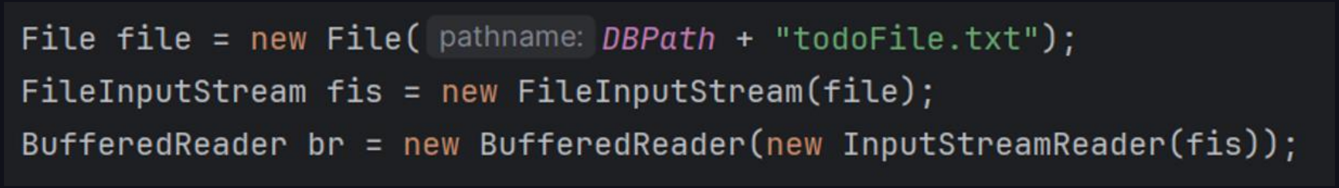
Create Example



```
    1 usage   new *
    public static void addTodoToStudent(String studentID, String taskName, String taskDate, String taskTime) {
        try {
            File file = new File( pathname: DBPath + "todoFile.txt");
            FileInputStream fis = new FileInputStream(file);
            BufferedReader br = new BufferedReader(new InputStreamReader(fis));

            String line;
            String[] info;
            StringBuilder sb =  new StringBuilder();

            while ((line = br.readLine()) != null) {
                info = line.split( regex: "\\$");
                if (info[0].equals(studentID)) {
                    sb.append(info[1]).append(",").append(taskName + "/" + taskDate + "/" + taskTime + "/1");
                    line = info[0] + "$" + sb.toString() + "$";
                }
                FileWriter fileWriter = new FileWriter( fileName: DBPath + "temp.txt",  append: true);
                fileWriter.write( str: line + '\n');
                fileWriter.close();
            }
            PrintWriter writer = new PrintWriter(file);
            writer.close();

            BufferedReader br2 = new BufferedReader(new FileReader( fileName: DBPath + "temp.txt"));
            while ((line = br2.readLine()) != null) {
                FileWriter fileWriter = new FileWriter( fileName: DBPath + "todoFile.txt",  append: true);
                fileWriter.write( str: line + '\n');
                fileWriter.close();
            }
            clearTempFile();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Let's walk through this code step by step to see how it works.

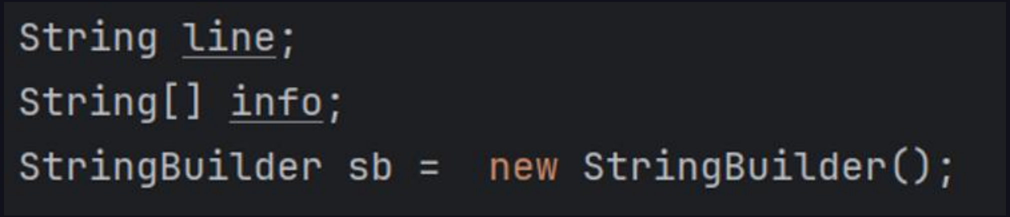
File Reading



```
File file = new File( pathname: DBPath + "todoFile.txt");
FileInputStream fis = new FileInputStream(file);
BufferedReader br = new BufferedReader(new InputStreamReader(fis));
```

- It starts by creating a File object for a file called todoFile.txt located at a specified DBPath.
- A BufferedReader is used to read from this file line by line.

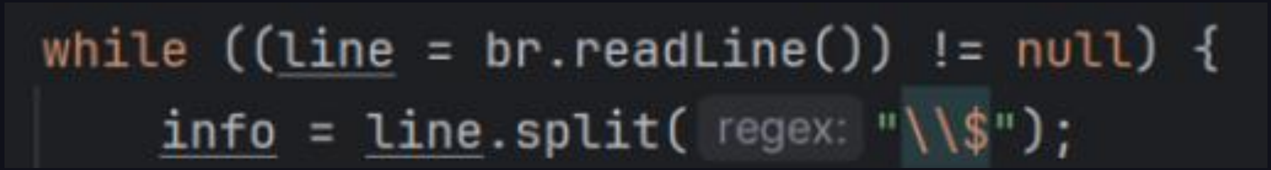
String Variables



```
String line;
String[] info;
StringBuilder sb =  new StringBuilder();
```

- String line; //To hold each line read from the file.
- String[] info; //An array to split the line into parts.
- StringBuilder sb; //To build the string that represents the updated task information for a particular student.

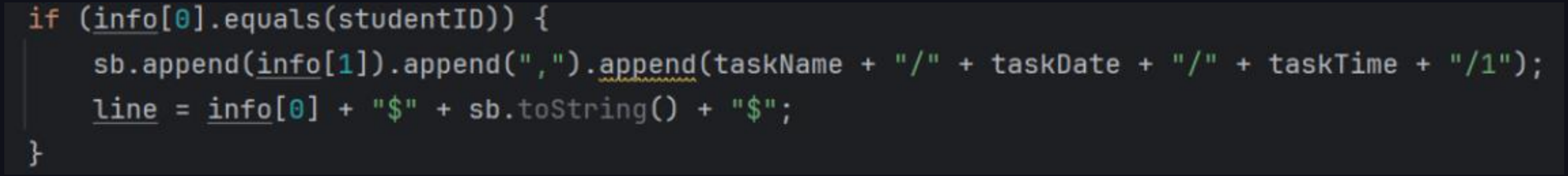
Reading the file line by line



```java
while ((line = br.readLine()) != null) {
    info = line.split( regex: "\\$");
```

- The file is read line by line. Each line is split by the delimiter "$" into the info array.
- The assumption is that the file stores the student's information in a format delimited by "$".

Checking and adding Task for the Specific Student:

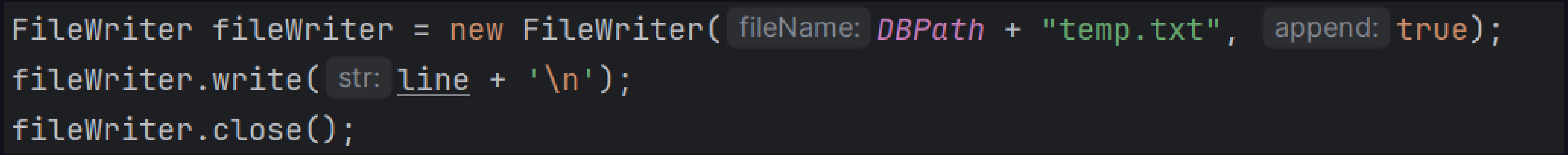


```
if (info[0].equals(studentID)) {
    sb.append(info[1]).append(",").append(taskName + "/" + taskDate + "/" + taskTime + "/1");
    line = info[0] + "$" + sb.toString() + "$";
}
```

If the student ID on the line matches the studentID provided as input:

- The second part of the line (which presumably stores tasks) is appended with the new task in the format: taskName/taskDate/taskTime/1.
- The line is updated to include the new task in the appropriate format.

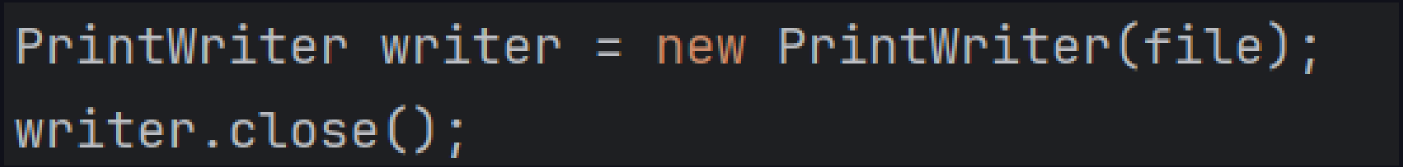
Writing to a temporary file



```java
FileWriter fileWriter = new FileWriter( fileName: DBPath + "temp.txt",  append: true);
fileWriter.write( str: line + '\n');
fileWriter.close();
```

After updating the line (or keeping it unchanged if it's not the targeted student), the method writes each line (either updated or unmodified) to a temporary file called temp.txt in the same directory (DBPath). This is done using a FileWriter, appending lines to avoid overwriting previous lines during the process.

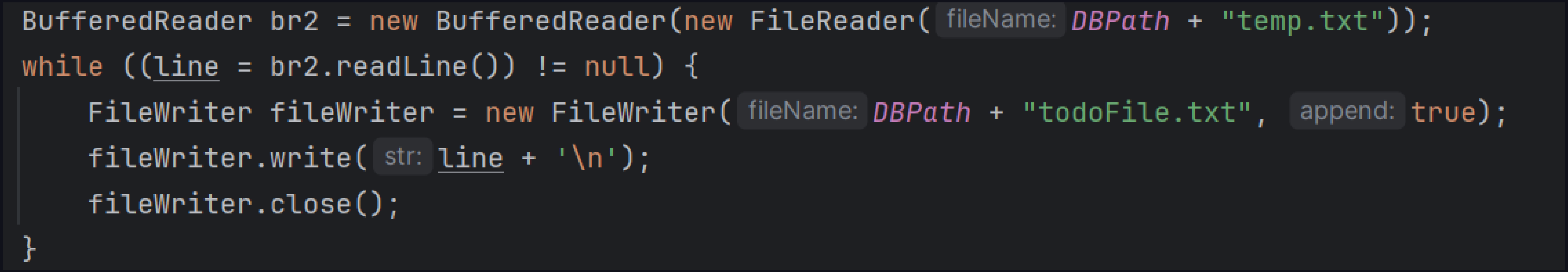
Replacing the original file



```
PrintWriter writer = new PrintWriter(file);
writer.close();
```

This part of the code clears the original todoFile.txt by opening it with a PrintWriter and immediately closing it, which effectively wipes the content of the file.

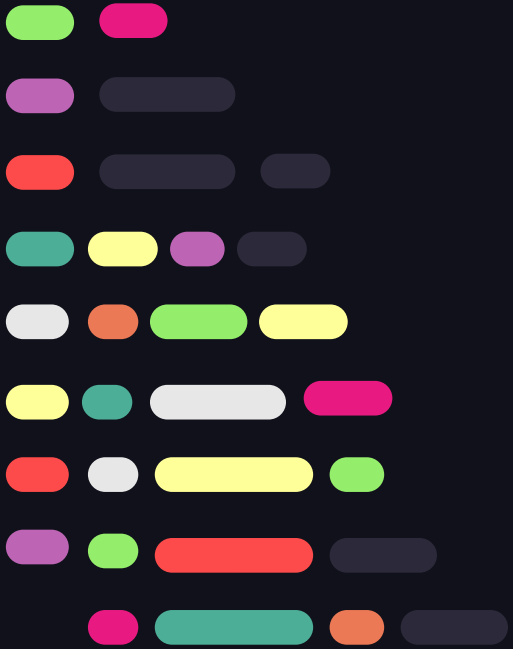
Copying back from the temp file



```
BufferedReader br2 = new BufferedReader(new FileReader( fileName: DBPath + "temp.txt"));
while ((line = br2.readLine()) != null) {
    FileWriter fileWriter = new FileWriter( fileName: DBPath + "todoFile.txt",  append: true);
    fileWriter.write( str: line + '\n');
    fileWriter.close();
}
```

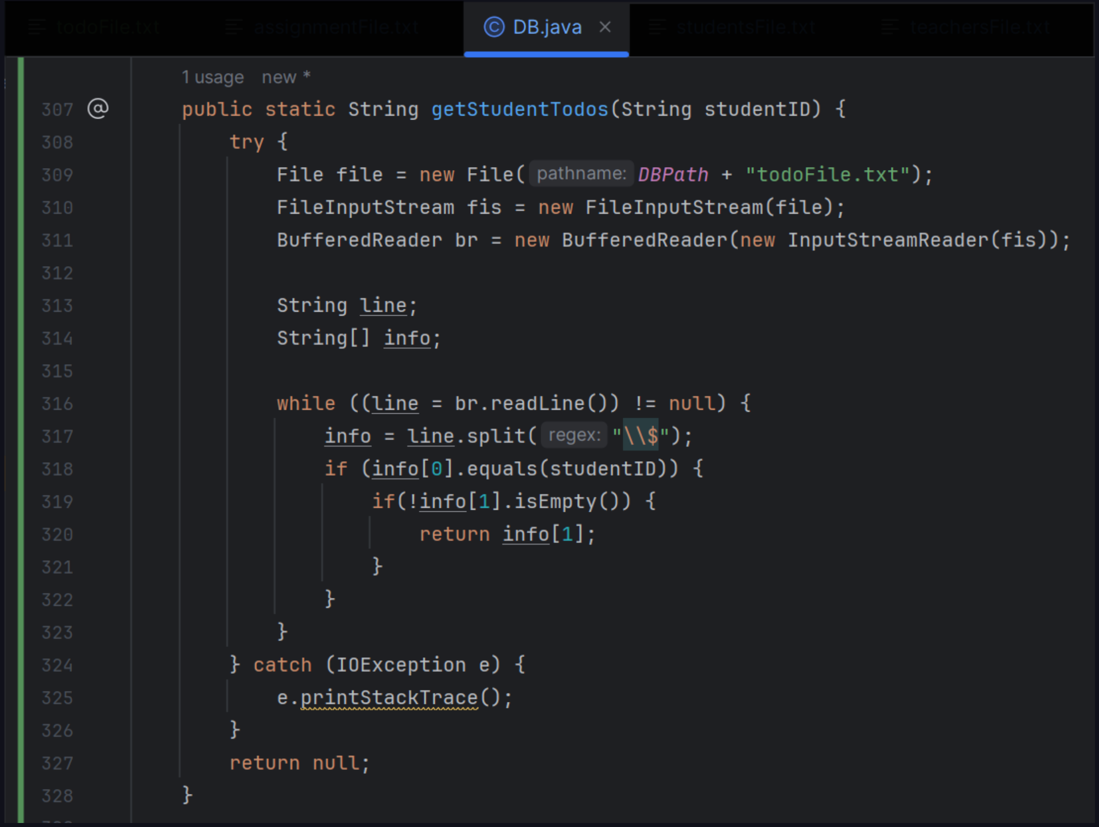
The method then reads the contents of the temp.txt file and writes it back into the original todoFile.txt. This ensures that the original file is updated with the new information for the student, and the temporary file is used only to handle the changes safely.

CRUD Example



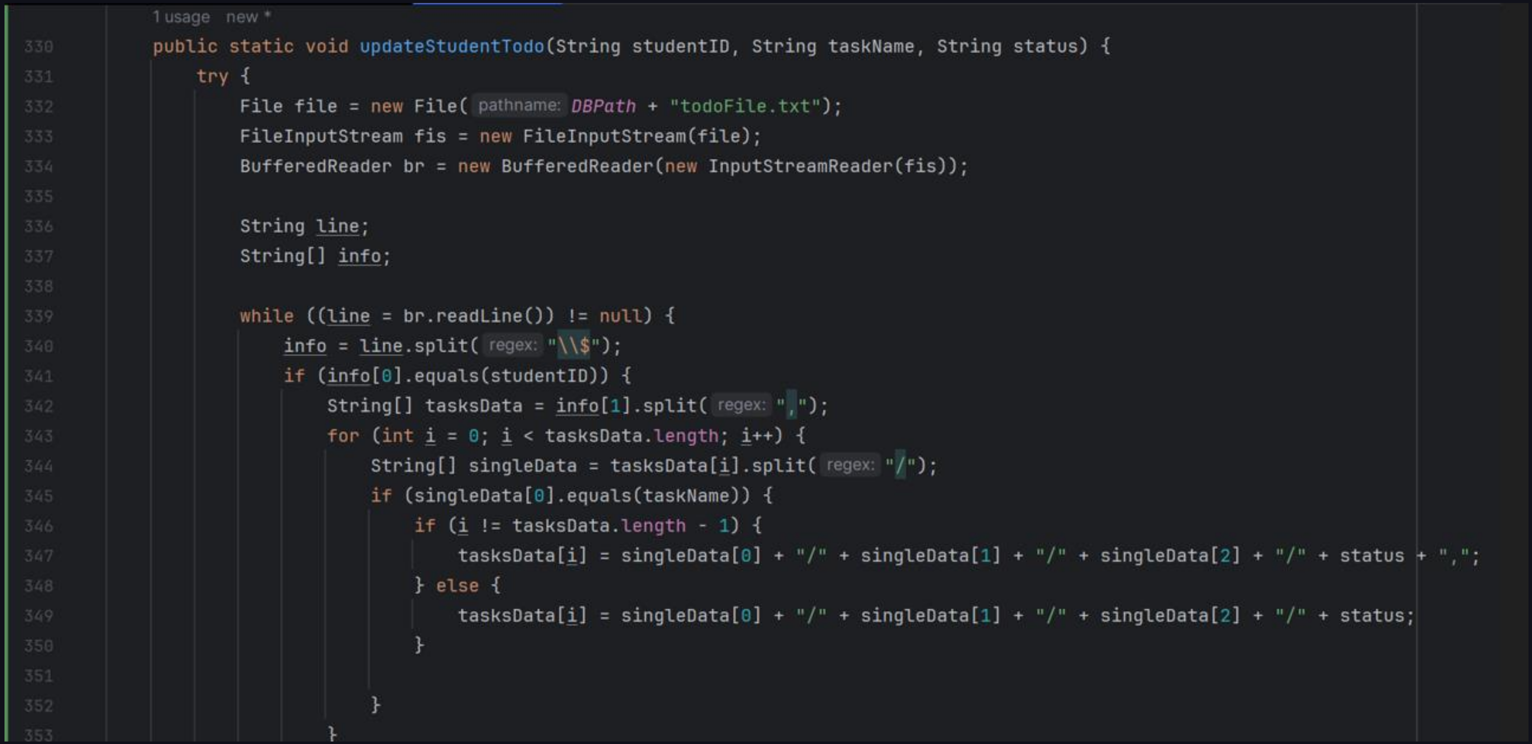
Now try to figure out how the rest of the code works based on the explanations for the create method (they are very similar!).

Read Example



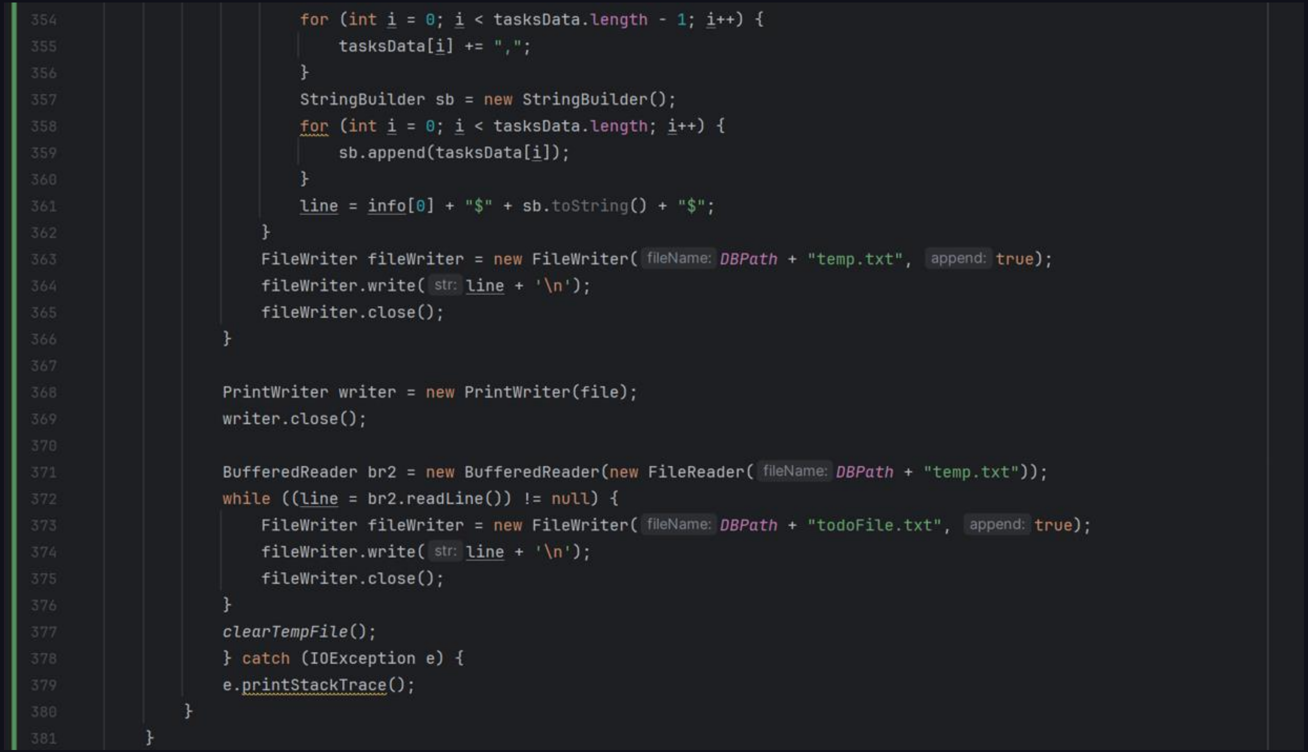
```
public static String getStudentTodos(String studentID) {
    try {
        File file = new File( pathname: DBPath + "todoFile.txt");
        FileInputStream fis = new FileInputStream(file);
        BufferedReader br = new BufferedReader(new InputStreamReader(fis));

        String line;
        String[] info;

        while ((line = br.readLine()) != null) {
            info = line.split( regex: "\\$");
            if (info[0].equals(studentID)) {
                if(!info[1].isEmpty()) {
                    return info[1];
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

Update Example



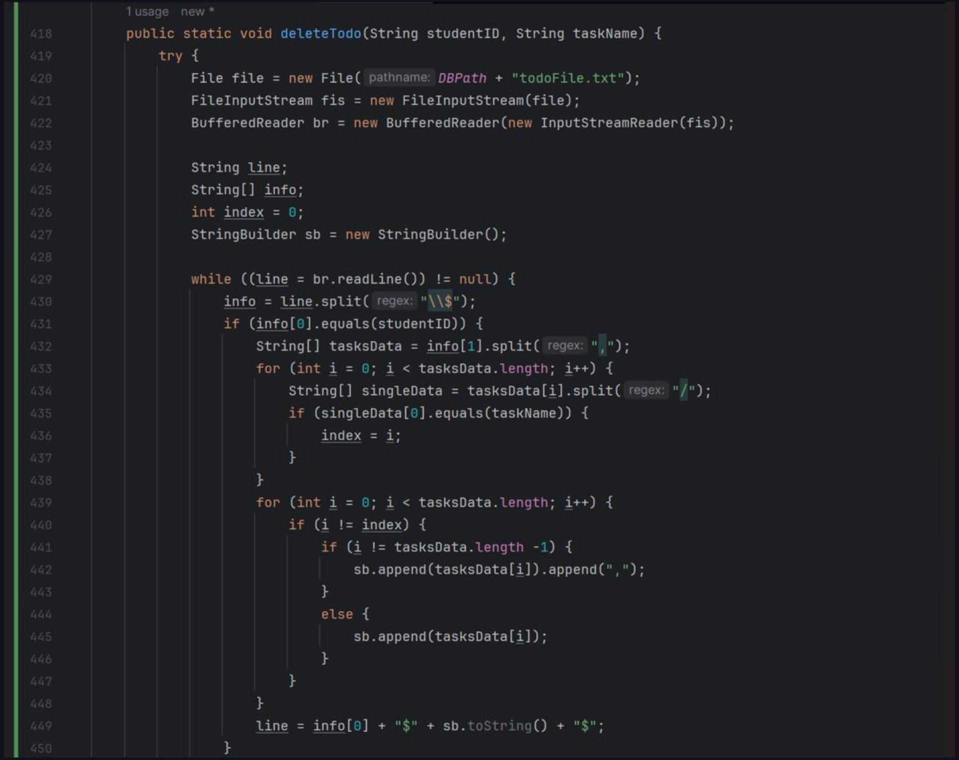
```
1 usage   new *
public static void updateStudentTodo(String studentID, String taskName, String status) {
    try {
        File file = new File( pathname: DBPath + "todoFile.txt");
        FileInputStream fis = new FileInputStream(file);
        BufferedReader br = new BufferedReader(new InputStreamReader(fis));

        String line;
        String[] info;

        while ((line = br.readLine()) != null) {
            info = line.split( regex: "\\$");
            if (info[0].equals(studentID)) {
                String[] tasksData = info[1].split( regex: ",");
                for (int i = 0; i < tasksData.length; i++) {
                    String[] singleData = tasksData[i].split( regex: "/");
                    if (singleData[0].equals(taskName)) {
                        if (i != tasksData.length - 1) {
                            tasksData[i] = singleData[0] + "/" + singleData[1] + "/" + singleData[2] + "/" + status + ",";
                        } else {
                            tasksData[i] = singleData[0] + "/" + singleData[1] + "/" + singleData[2] + "/" + status;
                        }

                    }
                }
```

Update Example



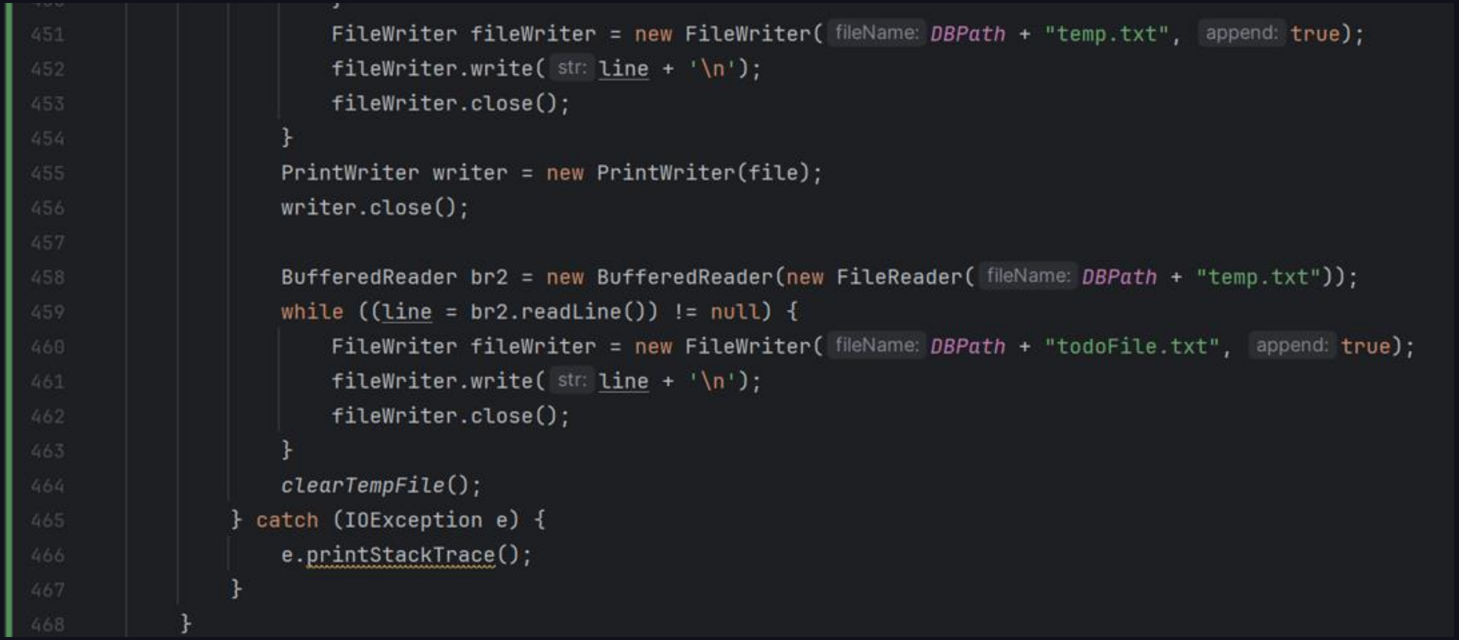
```java
                for (int i = 0; i < tasksData.length - 1; i++) {
                    tasksData[i] += ",";
                }
                StringBuilder sb = new StringBuilder();
                for (int i = 0; i < tasksData.length; i++) {
                    sb.append(tasksData[i]);
                }
                line = info[0] + "$" + sb.toString() + "$";
            }
            FileWriter fileWriter = new FileWriter( fileName: DBPath + "temp.txt",  append: true);
            fileWriter.write( str: line + '\n');
            fileWriter.close();
        }

        PrintWriter writer = new PrintWriter(file);
        writer.close();

        BufferedReader br2 = new BufferedReader(new FileReader( fileName: DBPath + "temp.txt"));
        while ((line = br2.readLine()) != null) {
            FileWriter fileWriter = new FileWriter( fileName: DBPath + "todoFile.txt",  append: true);
            fileWriter.write( str: line + '\n');
            fileWriter.close();
        }
        clearTempFile();
        } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Delete Example

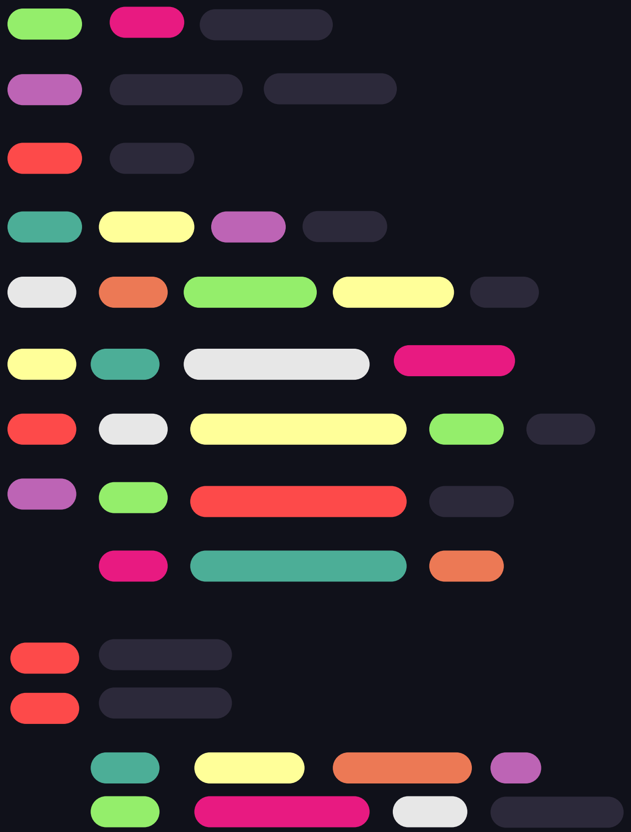


```
1 usage  new *
public static void deleteTodo(String studentID, String taskName) {
    try {
        File file = new File( pathname: DBPath + "todoFile.txt");
        FileInputStream fis = new FileInputStream(file);
        BufferedReader br = new BufferedReader(new InputStreamReader(fis));

        String line;
        String[] info;
        int index = 0;
        StringBuilder sb = new StringBuilder();

        while ((line = br.readLine()) != null) {
            info = line.split( regex: "\\$");
            if (info[0].equals(studentID)) {
                String[] tasksData = info[1].split( regex: ",");
                for (int i = 0; i < tasksData.length; i++) {
                    String[] singleData = tasksData[i].split( regex: "/");
                    if (singleData[0].equals(taskName)) {
                        index = i;
                    }
                }
                for (int i = 0; i < tasksData.length; i++) {
                    if (i != index) {
                        if (i != tasksData.length -1) {
                            sb.append(tasksData[i]).append(",");
                        }
                        else {
                            sb.append(tasksData[i]);
                        }
                    }
                }
                line = info[0] + "$" + sb.toString() + "$";
            }
```

Delete Example



```
                FileWriter fileWriter = new FileWriter( fileName: DBPath + "temp.txt",  append: true);
                fileWriter.write( str: line + '\n');
                fileWriter.close();
            }
            PrintWriter writer = new PrintWriter(file);
            writer.close();

            BufferedReader br2 = new BufferedReader(new FileReader( fileName: DBPath + "temp.txt"));
            while ((line = br2.readLine()) != null) {
                FileWriter fileWriter = new FileWriter( fileName: DBPath + "todoFile.txt",  append: true);
                fileWriter.write( str: line + '\n');
                fileWriter.close();
            }
            clearTempFile();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```



{ . .

Thanks for Tuning in

} . .