Example 8:

Implement an application that writes data to the SD Card.

External Storage

External storage such as SD card can also store application data, there’s no security enforced upon files you save to the external storage.
In general there are two types of External Storage:

- **Primary External Storage**: In built shared storage which is “accessible by the user by plugging in a USB cable and mounting it as a drive on a host computer

- **Secondary External Storage**: Removable storage. Example: SD Card

All applications can read and write files placed on the external storage and the user can remove them. We need to check if the SD card is available and if we can write to it.

The **FileInputStream** and **FileOutputStream** classes are used to read and write data into the file.

It is necessary to add external storage the permission to read and write. For that you need to add permission in android Manifest file.

Methods to Store Data in Android:

1. **getExternalStorageDirectory()** – Older way to access external storage in API Level less than 7. It is absolute now and not recommended. It directly get the reference to the root directory of your external storage or SD Card.

2. **getExternalFilesDir(String type) –** It is recommended way to enable us to create private files specific to app and files are removed as app is uninstalled. Example is app private data.

3. **getExternalStoragePublicDirectory() :** This is current recommended way that enable us to keep files public and are not deleted with the app uninstallation. Example images clicked by the camera exists even we uninstall the camera app.

FileInputStream:

This class reads the data from a specific file (byte by byte). It is usually used to read the contents of a file with raw bytes, such as images.

To read the contents of a file using this class −

First of all, you need to instantiate this class by passing a String variable or a File object, representing the path of the file to be read.

```
FileInputStream inputStream = new FileInputStream("file_path");
```

or,

```
File file = new File("file_path");
FileInputStream inputStream = new FileInputStream(file);
```

Then read the contents of the specified file using either of the variants of read() method −

- **int read()** − This simply reads data from the current InputStream and returns the read data byte by byte (in integer format).

  This method returns -1 if the end of the file is reached.

- **int read(byte[] b)** − This method accepts a byte array as parameter and reads the contents of the current InputStream, to the given array

  This method returns an integer representing the total number of bytes or, -1 if the end of the file is reached.

- **int read(byte[] b, int off, int len)** − This method accepts a byte array, its offset (int) and, its length (int) as parameters and reads the contents of the current InputStream, to the given array.

  This method returns an integer representing the total number of bytes or, -1 if the end of the file is reached.

FileOutputStream:

This writes data into a specific file or, file descriptor (byte by byte). It is usually used to write the contents of a file with raw bytes, such as images.

To write the contents of a file using this class −

First of all, you need to instantiate this class by passing a String variable or a **File** object, representing the path of the file to be read.

```
FileOutputStream outputStream = new FileOutputStream("file_path");
```

or,

```
File file = new File("file_path");

FileOutputStream outputStream = new FileOutputStream (file);
```

You can also instantiate a FileOutputStream class by passing a FileDescriptor object.

```
FileDescriptor descriptor = new FileDescriptor();

FileOutputStream outputStream = new FileOutputStream(descriptor);
```

Then write the data to a specified file using either of the variants of write() method −

- **int write(int b)** − This method accepts a single byte and writes it to the current OutputStream.

- **int write(byte[] b)** − This method accepts a byte array as parameter and writes data from it to the current OutputStream.

- **int write(byte[] b, int off, int len)** − This method accepts a byte array, its offset (int) and, its length (int) as parameters and writes its contents to the current OutputStream.