Name – Pawan Kumar

Prn No – 22030121149

Div – A

Q1.

Ans: **External storage** in Android refers to storage media that is outside the internal memory of a device but still accessible to the system. It can be physical (such as an SD card) or virtual (a partition within the internal memory that is accessible as external).

**1. Key Differences Between External Storage and Internal Storage in Android:**

* **Location**:
  + **Internal storage** is dedicated space on the device where app data and system files reside. Users and other apps generally cannot access these files.
  + **External storage** is a shared space that apps, users, and the system can access more freely. It may be removable (like an SD card) or non-removable but still considered "external" (like a partition in the internal memory).
* **Accessibility**:
  + **Internal storage** is private to the application unless the user has root access.
  + **External storage** can be accessed by any app with the necessary permissions. This makes it more prone to corruption or accidental deletion.
* **Persistence**:
  + Data in **internal storage** is persistent, meaning it is not affected when the app is uninstalled.
  + Data in **external storage** can be removed when the user uninstalls an app, especially if the app specifically stores data in public folders.
* **Security**:
  + **Internal storage** provides higher security as it is private and restricted to the app.
  + **External storage** is less secure, especially on devices with removable SD cards, as anyone with physical access to the card can view the data.

**2. Availability and Accessibility of External Storage Across Devices:**

* **Device Variability**:
  + The availability of external storage can differ across Android devices. Some devices may not support physical SD cards, while others provide this option. Devices with virtual external storage typically partition a part of the internal storage and present it as external storage.
* **User Permissions**:
  + External storage access is often governed by runtime permissions. Starting from Android 6.0 (API level 23), users are prompted to grant apps permission to read or write to external storage.
* **File Organization**:
  + Files in external storage can be categorized into **public** and **private** directories. Public directories (like "Downloads" or "Pictures") are accessible to all apps and users, while private directories, created by apps, can only be accessed by the app that owns them.
* **Behavior on Different Android Versions**:
  + Android versions since 10 (API level 29) implement **Scoped Storage**, which changes how apps can access files on external storage. Apps are now sandboxed in external storage, meaning they only have access to their own directories unless explicit permission is granted.

--------------------------------------------------------------------------------------------------------------------------------------

Q2.

Ans:**1. Evolution of the Permission Model (Android 6.0 to the Latest Version)**

* **Android 6.0 (Marshmallow) – Runtime Permissions Introduced**:
  + Before Android 6.0, permissions were granted at install time, meaning users had to accept all permissions an app requested before installation.
  + Starting from **Android 6.0**, the concept of **runtime permissions** was introduced. This means apps must request permissions dynamically, during usage, rather than at installation. The user can approve or deny access to external storage at runtime.
  + For external storage, the specific permissions are READ\_EXTERNAL\_STORAGE and WRITE\_EXTERNAL\_STORAGE, which allow reading and writing to external storage.
* **Android 10 (Scoped Storage)**:
  + Android 10 introduced **Scoped Storage**, which provides apps with access only to their own files in external storage (sandboxed space). Access to other files or public directories requires explicit permission from the user.
  + The goal was to protect user privacy by preventing apps from indiscriminately accessing other apps' or user data.
  + Even with the READ\_EXTERNAL\_STORAGE permission, apps cannot access files in other apps' directories.
* **Android 11**:
  + The permission model introduced **All Files Access** (MANAGE\_EXTERNAL\_STORAGE) for apps that need broader access to external storage (e.g., file managers).
  + Apps requesting this permission must justify its usage, and the permission is strictly controlled and reviewed.
* **Android 13**:
  + Android 13 introduces more granular permissions such as READ\_MEDIA\_IMAGES, READ\_MEDIA\_VIDEO, and READ\_MEDIA\_AUDIO, reducing the scope of external storage permissions and allowing apps to request only the necessary access to specific types of media files.
  + This enhances security and gives users more control over which files apps can access.

**2. Key Considerations When Requesting Permissions Dynamically at Runtime**

* **User Experience**:
  + When requesting permissions, it's important to explain clearly why the permission is needed. This can be done through a rationale message before prompting the user.
  + For example, if an app requests access to photos or videos, it should explain how it will use those files to provide better clarity and increase user trust.
* **Granularity**:
  + Request only the permissions necessary for the app's core functionality. Over-requesting permissions may result in user denial and cause the app to lose credibility.
  + Since Android 13, apps can ask for media-specific permissions instead of blanket external storage access. This granularity can reduce the burden on users to approve unnecessary access.
* **Handling Denials**:
  + Apps need to handle the case where a user denies permission. Provide alternative ways for the user to use the app or guide them to settings if the feature is critical.
* **Permissions Best Practices**:
  + Always check if the permission has already been granted before requesting it.
  + Handle both permission grant and denial scenarios gracefully without breaking the user experience.
  + Use intents when possible for file access, which might reduce the need for external storage permissions.

--------------------------------------------------------------------------------------------------------------------------------------

Q3.

Ans: **Scoped Storage** is a storage model introduced in **Android 10** to enhance user privacy and data security by restricting how apps can access external storage.

**1. Motivations Behind the Introduction of Scoped Storage:**

* **User Privacy**:
  + Prior to Android 10, apps with the READ\_EXTERNAL\_STORAGE or WRITE\_EXTERNAL\_STORAGE permission could access the entire external storage, including other apps' data, which posed a significant privacy risk. There was no restriction on reading or writing to any folder in external storage, which could lead to data misuse or leakage.
* **Data Security**:
  + Scoped Storage was introduced to prevent unauthorized or malicious apps from accessing sensitive user data. By isolating app data and giving each app its own sandboxed directory in external storage, Scoped Storage significantly improves the security of user data.
* **Storage Performance**:
  + Another motivation was to improve file organization and storage performance by restricting the number of apps with broad access to external storage. Uncontrolled access often led to clutter and data management issues.

**2. How Scoped Storage Improves Security and Privacy Compared to the Previous Model:**

* **App-Specific Storage**:
  + Under Scoped Storage, each app has its own dedicated directory in external storage. This app-specific directory is sandboxed, meaning only the app that owns it can read or write to it. Other apps cannot access this data, even if they have storage permissions.
* **Restricted Permissions**:
  + In the Scoped Storage model, even if an app is granted the READ\_EXTERNAL\_STORAGE or WRITE\_EXTERNAL\_STORAGE permission, it does not have access to other apps' files or public directories like before. It can only access files that it owns or files explicitly shared by the user.
* **Granular Access**:
  + Apps that need access to shared or public files (e.g., photos, videos, documents) must request specific access from users. The introduction of **media-specific permissions** in Android 13, such as READ\_MEDIA\_IMAGES or READ\_MEDIA\_VIDEO, allows apps to request access only to certain types of files without requiring broad storage permissions.
* **Improved File Access**:
  + Scoped Storage also allows apps to interact with files without needing to access the actual file paths. The use of **Storage Access Framework (SAF)** and **MediaStore API** enables apps to interact with documents, images, and other media while maintaining user privacy and system stability.
* **All Files Access Exception**:
  + For apps like file managers that need broad access to external storage, Android provides an exception called **All Files Access** (MANAGE\_EXTERNAL\_STORAGE). However, this permission is highly restricted, and developers need to justify its usage, further enhancing overall security.

--------------------------------------------------------------------------------------------------------------------------------------

Q4.

Ans: To handle external storage permissions in an Android app, developers need to follow these general steps:

**1. Declare Permissions in the Manifest:**

* Start by declaring the necessary permissions in the app’s AndroidManifest.xml file. For external storage, permissions like reading and writing must be declared. However, newer Android versions encourage using scoped storage to minimize the need for broad external storage access.

**2. Request Runtime Permissions:**

* For Android 6.0 (API level 23) and above, storage permissions are considered dangerous, meaning users must explicitly grant them at runtime. The app should first check if it already has permission, and if not, prompt the user with a permission request dialog.

**3. Handle the User’s Response:**

* When the user responds to the permission request, the app must handle their response. If permission is granted, the app can proceed with accessing external storage. If the user denies the request, the app should gracefully handle the denial by disabling features that depend on storage access.

**4. Provide a Fallback Mechanism:**

* If the user denies permission, it's important to explain why the permission is needed before asking again. Android provides a way to determine if the user might need more information about why the permission is necessary.
* If the user denies the permission permanently (e.g., selecting "Don’t ask again"), the app should provide a fallback mechanism. This might include limiting functionality or prompting users to enable permissions via the app’s settings.

**5. Adapt to Scoped Storage (Android 10 and Above):**

* With Android 10, the need for broad storage permissions has reduced due to scoped storage, which restricts access to shared storage and provides each app with its own isolated space. When working with shared files (like images or videos), developers should consider using Android's MediaStore API or Storage Access Framework, which provide more secure and limited access to external files.

--------------------------------------------------------------------------------------------------------------------------------------

Q5.

Ans: **Comparison of Storage Options in Android:**

**1. Internal Storage:**

* **Pros:**
  + **Security:** Data stored in internal storage is private to the app and cannot be accessed by other apps or users.
  + **Automatic Cleanup:** Data is automatically removed when the app is uninstalled, reducing the need for manual cleanup.
  + **No Permission Needed:** Accessing internal storage doesn’t require explicit user permission, simplifying usage.
* **Cons:**
  + **Limited Space:** Internal storage is typically limited, so storing large files (like multimedia) is not ideal.
  + **Device-Dependent:** Data is stored locally on the device, so it’s inaccessible if the user changes devices or uninstalls the app.
* **Ideal Scenarios:**
  + Sensitive user data (such as login credentials).
  + Small files that don’t require sharing with other apps.

**2. External Storage:**

* **Pros:**
  + **Large Capacity:** External storage (such as SD cards) usually offers more space, allowing for the storage of larger files like photos, videos, and downloads.
  + **Accessible to Other Apps:** Files stored here can be shared with other apps or users.
* **Cons:**
  + **Security Risk:** Files are accessible to other apps, making it less secure.
  + **Permission Required:** Starting from Android 6.0, explicit permissions from the user are required to access external storage.
  + **No Automatic Cleanup:** Files remain even after the app is uninstalled, which can lead to clutter.
* **Ideal Scenarios:**
  + Storing large media files (e.g., images, videos) or documents.
  + Sharing files between apps.

**3. Cloud-Based Storage:**

* **Pros:**
  + **Accessibility:** Data is accessible from multiple devices, enabling backup and syncing across platforms.
  + **No Local Storage Limits:** Since data is stored online, it doesn’t consume device storage, ideal for apps with large data requirements.
  + **Security (Depends on Provider):** Most cloud services provide secure encryption and user authentication.
* **Cons:**
  + **Internet Dependency:** Requires an active internet connection to access or sync files.
  + **Potential Privacy Concerns:** Storing sensitive data in the cloud might raise privacy or security concerns.
  + **Cost:** Storing large amounts of data may incur additional costs, depending on the provider.
* **Ideal Scenarios:**
  + Backing up app data, user preferences, and media files.
  + Apps that need to sync data across multiple devices or platforms.