BỘ NÔNG NGHIỆP VÀ MÔI TRƯỜNG
TRƯỜNG ĐẠI HỌC THỦY LỢI

BÀI TẬP THỰC HÀNH SỐ 5

PHÁT TRIỂN ỨNG DỤNG CHO THIẾT BỊ DI ĐỘNG

NỘI DUNG BỔ SUNG: ỨNG DỤNG VỚI CHỦ ĐỀ NÂNG CAO

STT	Mã sinh viên	Họ và tên	Lớp
1	2251061731	Lường Văn Cương	64CNTT2

Hà Nội, năm 2025

BÀI TẬP 1: Content Providers

Mục tiêu:

- Tìm hiểu cách sử dụng Content Providers để truy cập dữ liệu từ ứng dụng khác (ứng dụng Danh bạ).
- Hiển thị danh sách tên các liên hệ trong danh bạ lên ứng dụng của mình.

Các bước thực hiện:

1. **Thiết lập quyền truy cập:**
   - Mở file `AndroidManifest.xml` của ứng dụng.
   - Thêm quyền `READ_CONTACTS` để xin phép ứng dụng được đọc dữ liệu danh bạ.

   XML

   ```
   <uses-permission android:name="android.permission.READ_CONTACTS" />
   ```

2. **Thiết kế giao diện (layout):**
   - Tạo một `ListView` trong file layout (ví dụ: `activity_main.xml`) để hiển thị danh sách tên liên hệ.

   XML

   ```
   <ListView
       android:id="@+id/listViewContacts"
       android:layout_width="match_parent"
       android:layout_height="match_parent" />
   ```

3. **Đọc dữ liệu từ Content Provider:**
   - Trong Activity chính (ví dụ: `MainActivity.kt`), sử dụng `ContentResolver` để truy vấn dữ liệu từ Content Provider của ứng dụng Danh bạ.
   - URI của Content Provider Danh bạ là: `ContactsContract.Contacts.CONTENT_URI`
   - Sử dụng phương thức `query()` của `ContentResolver` để lấy dữ liệu. Phương thức này trả về một `Cursor` chứa kết quả truy vấn.
   - Duyệt `Cursor` để lấy tên của từng liên hệ và lưu vào một `ArrayList<String>`.
4. **Hiển thị dữ liệu lên ListView:**
   - Tạo một `ArrayAdapter<String>` để đưa dữ liệu từ `ArrayList<String>` lên `ListView`.
   - Gán `ArrayAdapter<String>` cho `ListView`.

Code ví dụ (MainActivity.kt):

Kotlin

```
import android.Manifest
import android.content.pm.PackageManager
import android.database.Cursor
import android.os.Bundle
import android.provider.ContactsContract
import android.widget.ArrayAdapter
import android.widget.ListView
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
```

```
class MainActivity : AppCompatActivity() {

    private lateinit var listViewContacts: ListView
    private lateinit var contactsList: ArrayList<String>
    private lateinit var contactsAdapter: ArrayAdapter<String>

    private val REQUEST_READ_CONTACTS_PERMISSION = 100

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        listViewContacts = findViewById(R.id.listViewContacts)
        contactsList = ArrayList()

        // Kiểm tra và xin quyền READ_CONTACTS
        if (ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.READ_CONTACTS
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            ActivityCompat.requestPermissions(
                this,
                arrayOf(Manifest.permission.READ_CONTACTS),
                REQUEST_READ_CONTACTS_PERMISSION
            )
        } else {
            loadContacts()
        }
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults)
        if (requestCode == REQUEST_READ_CONTACTS_PERMISSION) {
            if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                loadContacts()
            } else {
                // Xử lý trường hợp người dùng từ chối cấp quyền
                // Ví dụ: Hiển thị thông báo cho người dùng
                // Giải thích lý do cần quyền truy cập danh bạ
                Toast.makeText(this, "Permission denied",
Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun loadContacts() {
        // Lấy dữ liệu từ Content Provider
        val cursor: Cursor? = contentResolver.query(
            ContactsContract.Contacts.CONTENT_URI,
            null,
            null,
            null,
            null
        )

        cursor?.use {
```

```
            if (it.count > 0) {
                while (it.moveToNext()) {
                    val nameIndex =
it.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)
                    val name = it.getString(nameIndex)
                    contactsList.add(name)
                }
            }
        }

        // Hiển thị dữ liệu lên ListView
        contactsAdapter = ArrayAdapter(this,
android.R.layout.simple_list_item_1, contactsList)
        listViewContacts.adapter = contactsAdapter
    }
}
```

Giải thích chi tiết (Kotlin):

- `Manifest.permission.READ_CONTACTS`: Khai báo quyền truy cập danh bạ trong `AndroidManifest.xml`.
- `ContextCompat.checkSelfPermission()`: Kiểm tra xem ứng dụng đã được cấp quyền `READ_CONTACTS` hay chưa.
- `ActivityCompat.requestPermissions()`: Xin quyền `READ_CONTACTS` từ người dùng nếu chưa được cấp.
- `onRequestPermissionsResult()`: Xử lý kết quả trả về khi người dùng cấp hoặc từ chối quyền.
- `contentResolver`: Đối tượng `ContentResolver` cho phép ứng dụng tương tác với Content Providers.
- `ContactsContract.Contacts.CONTENT_URI`: URI của Content Provider chứa dữ liệu về các liên hệ.
- `Cursor`: Một interface đại diện cho tập kết quả của một truy vấn cơ sở dữ liệu. Trong trường hợp này, nó chứa dữ liệu từ Content Provider.
- `cursor?.use {}`: Sử dụng `use` block để tự động đóng `Cursor` sau khi sử dụng, tránh rò rỉ tài nguyên.
- `it.count`: Lấy số lượng hàng trong `Cursor`.
- `it.moveToNext()`: Di chuyển đến hàng tiếp theo trong `Cursor`.
- `it.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)`: Lấy chỉ số của cột chứa tên hiển thị của liên hệ.
- `it.getString(nameIndex)`: Lấy giá trị kiểu String từ cột tại chỉ số đã cho.
- `ArrayAdapter<String>`: Adapter để hiển thị danh sách các chuỗi (tên liên hệ) lên `ListView`.
- `listViewContacts.adapter`: Gán `ArrayAdapter` cho `ListView` để hiển thị dữ liệu.

Hướng dẫn Bài tập 01*: Chụp lại mà hình từng bước thực hiện (tương tự cho các Bài tập bên dưới) để học sinh lớp 10 có thể thực hiện lại theo được :D*

BÀI TẬP 2: Ứng dụng tự động trả lời tin nhắn cuộc gọi nhỡ

Mục tiêu:

- Sử dụng Broadcast Receiver để lắng nghe sự kiện cuộc gọi đến.
- Sử dụng Telephony API để lấy thông tin về cuộc gọi (số điện thoại).
- Sử dụng SMS API để gửi tin nhắn SMS.

Mô tả:

Ứng dụng sẽ tự động gửi một tin nhắn SMS đến số điện thoại của người gọi nhỡ, với nội dung thông báo rằng bạn đang bận và sẽ gọi lại sau.

Các bước thực hiện:

1. **Khai báo quyền:**
   - Thêm các quyền cần thiết vào `AndroidManifest.xml`:
     - `android.permission.READ_PHONE_STATE` (để theo dõi trạng thái cuộc gọi)
     - `android.permission.SEND_SMS` (để gửi tin nhắn SMS)
     - `android.permission.RECEIVE_SMS` (nếu muốn xử lý cả tin nhắn đến)
2. **Tạo Broadcast Receiver:**
   - Tạo một class kế thừa `BroadcastReceiver` để lắng nghe sự kiện `android.intent.action.PHONE_STATE`.
   - Trong phương thức `onReceive()`:
     - Kiểm tra trạng thái cuộc gọi (`TelephonyManager.EXTRA_STATE_RINGING`).
     - Nếu là cuộc gọi đến, lấy số điện thoại người gọi (`intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER)`).
     - Nếu cuộc gọi bị nhỡ (có thể theo dõi thêm sự kiện `TelephonyManager.CALL_STATE_IDLE` sau khi đổ chuông), gửi tin nhắn SMS đến số điện thoại đó.
3. **Gửi tin nhắn SMS:**
   - Sử dụng `SmsManager` để gửi tin nhắn SMS.
   - `SmsManager.getDefault().sendTextMessage()` để gửi tin nhắn.
4. **Đăng ký Broadcast Receiver:**
   - Đăng ký receiver trong `AndroidManifest.xml`.

Hướng dẫn Bài tập 02: *Chụp lại mà hình từng bước thực hiện (tương tự cho các Bài tập bên dưới) để học sinh lớp 10 có thể thực hiện lại theo được :D*

BÀI TẬP 3: Ứng dụng chặn cuộc gọi theo số điện thoại

Mục tiêu:

- Sử dụng Broadcast Receiver để lắng nghe sự kiện cuộc gọi đến.
- Sử dụng Telephony API để lấy thông tin về cuộc gọi (số điện thoại).
- (Nâng cao) Tìm hiểu cách chặn cuộc gọi (có thể cần các phương pháp không chính thức hoặc API riêng của nhà sản xuất điện thoại).

Mô tả:

Ứng dụng sẽ tự động từ chối hoặc ngắt kết nối các cuộc gọi đến từ một danh sách các số điện thoại bị chặn.

Các bước thực hiện:

1. **Khai báo quyền:**
   - Thêm quyền `android.permission.READ_PHONE_STATE` vào `AndroidManifest.xml`.
   - (Có thể cần thêm các quyền liên quan đến xử lý cuộc gọi tùy theo phương pháp chặn)
2. **Tạo Broadcast Receiver:**
   - Tạo một class kế thừa `BroadcastReceiver` để lắng nghe sự kiện `android.intent.action.PHONE_STATE`.
   - Trong phương thức `onReceive()`:
     - Kiểm tra trạng thái cuộc gọi (`TelephonyManager.EXTRA_STATE_RINGING`).
     - Nếu là cuộc gọi đến, lấy số điện thoại người gọi.
     - Kiểm tra xem số điện thoại đó có nằm trong danh sách chặn hay không.
     - Nếu có trong danh sách chặn, thực hiện hành động chặn cuộc gọi.
3. **Chặn cuộc gọi:**
   - (Phần này có thể phức tạp và phụ thuộc vào phiên bản Android và nhà sản xuất điện thoại)
   - Có thể cần sử dụng `TelephonyManager` hoặc các API khác để thực hiện chặn cuộc gọi.
   - Lưu ý rằng việc chặn cuộc gọi có thể bị hạn chế hoặc không được hỗ trợ trên một số thiết bị.
4. **Đăng ký Broadcast Receiver:**
   - Đăng ký receiver trong `AndroidManifest.xml`.

Lưu ý quan trọng:

- **Xử lý bất đồng bộ trong `onReceive()`:** Tránh thực hiện các tác vụ tốn thời gian trong phương thức `onReceive()` của Broadcast Receiver. Nếu cần thực hiện các tác vụ dài, hãy sử dụng Service.
- **Quyền (Permissions):** Các thao tác liên quan đến Telephony và SMS đều yêu cầu các quyền đặc biệt. Đảm bảo bạn đã khai báo đầy đủ các quyền trong `AndroidManifest.xml` và xử lý việc xin quyền từ người dùng một cách thích hợp (đặc biệt là trên các phiên bản Android mới).
- **Hạn chế của Telephony API:** Một số chức năng liên quan đến Telephony có thể bị hạn chế hoặc không được hỗ trợ trên một số thiết bị hoặc phiên bản Android.

- **SMS PDU:** Khi nhận SMS, dữ liệu thường ở định dạng PDU. Cần xử lý để giải mã và đọc nội dung tin nhắn.

BÀI TẬP 4: Ứng dụng tải và hiển thị ảnh từ Internet

Mục tiêu:

- Sử dụng `AsyncTask` để thực hiện tải ảnh từ một URL trên Internet trong background.
- Hiển thị ảnh đã tải xuống lên ImageView trong UI Thread.
- Hiển thị progress bar trong khi tải ảnh.

Mô tả:

Ứng dụng cho phép người dùng nhập một URL ảnh. Sau khi người dùng nhấn nút, ứng dụng sẽ hiển thị một progress bar và bắt đầu tải ảnh từ URL đó trong background. Khi tải xong, ứng dụng sẽ ẩn progress bar và hiển thị ảnh lên ImageView.

Các bước thực hiện:

1. **Thiết kế giao diện:**
    - Một EditText để người dùng nhập URL.
    - Một ImageView để hiển thị ảnh.
    - Một ProgressBar để hiển thị tiến trình tải.
    - Một Button để kích hoạt quá trình tải.
2. **Tạo AsyncTask:**
    - Tạo một class kế thừa `AsyncTask<String, Integer, Bitmap>`.
        - `String`: URL của ảnh.
        - `Integer`: Tiến trình tải (ví dụ: phần trăm hoàn thành).
        - `Bitmap`: Ảnh đã tải.
    - Implement các phương thức:
        - `onPreExecute()`: Hiển thị progress bar.
        - `doInBackground(String... urls)`:
            - Tải ảnh từ URL (sử dụng các thư viện như `HttpURLConnection` hoặc `OkHttp`).
            - Trong quá trình tải, gọi `publishProgress()` để cập nhật tiến trình (ví dụ: phần trăm tải).
            - Trả về `Bitmap` của ảnh đã tải.
        - `onProgressUpdate(Integer... values)`: Cập nhật progress bar trên UI thread.
        - `onPostExecute(Bitmap result)`:
            - Ẩn progress bar.
            - Hiển thị ảnh lên ImageView (nếu tải thành công).
3. **Xử lý sự kiện Button click:**
    - Khi người dùng click nút, lấy URL từ EditText.
    - Tạo một instance của `AsyncTask` và gọi `execute(url)`.

BÀI TẬP 5: Ứng dụng đếm giờ và cập nhật giao diện

Mục tiêu:

- Sử dụng `Handler` và `Runnable` để cập nhật UI định kỳ từ một thread khác.
- Hiển thị thời gian đã trôi qua trên TextView.

Mô tả:

Ứng dụng hiển thị một TextView để hiển thị thời gian đã trôi qua (ví dụ: số giây). Một thread nền sẽ tăng giá trị thời gian và sử dụng Handler để gửi thông tin cập nhật lên UI thread để hiển thị.

Các bước thực hiện:

1. **Thiết kế giao diện:**
    - Một TextView để hiển thị thời gian.
2. **Tạo Handler:**
    - Tạo một `Handler` trong Activity chính.
    - Override phương thức `handleMessage()` (nếu dùng Message) hoặc tạo một `Runnable`.
3. **Tạo Thread:**
    - Tạo một Thread mới.
    - Trong `run()` method:
        - Lặp lại việc tăng giá trị thời gian.
        - Sử dụng `Handler.post(runnable)` để gửi một `Runnable` lên UI thread để cập nhật TextView.
4. **Cập nhật TextView:**
    - Trong `Runnable`, cập nhật `TextView.setText()` với giá trị thời gian hiện tại.

Lưu ý:

- **UI Thread:** Chỉ có UI thread mới được phép trực tiếp cập nhật các thành phần giao diện người dùng.
- **Tránh các tác vụ dài trên UI Thread:** Các tác vụ tốn thời gian (ví dụ: tải dữ liệu mạng, xử lý ảnh) nên được thực hiện trong background thread để tránh làm treo ứng dụng.
- **Sử dụng AsyncTask cho các tác vụ đơn giản:** `AsyncTask` là một cách dễ dàng để thực hiện các tác vụ nền đơn giản và tương tác với UI thread.
- **Sử dụng Handler và Thread cho các tác vụ phức tạp hơn:** `Handler` và `Thread` cung cấp sự linh hoạt cao hơn cho các tác vụ nền phức tạp hoặc cần kiểm soát thread chi tiết hơn.

BÀI TẬP 6: Ứng dụng ghi âm và phát lại

Mục tiêu:

- Sử dụng `MediaRecorder` để ghi âm từ microphone của thiết bị.
- Lưu file ghi âm vào MediaStore để các ứng dụng khác có thể truy cập.
- Sử dụng `MediaPlayer` để phát lại file ghi âm.
- Hiển thị danh sách các file ghi âm đã lưu trong MediaStore.

Mô tả:

Ứng dụng cho phép người dùng ghi âm, xem danh sách các bản ghi đã thực hiện và phát lại chúng.

Các bước thực hiện:

1. **Ghi âm:**
    - Sử dụng `MediaRecorder` để ghi âm.
    - Thiết lập các thông số cần thiết như nguồn âm thanh, định dạng file, nơi lưu trữ.
    - Lưu file ghi âm vào một vị trí cụ thể.
    - Sử dụng `ContentValues` để thêm thông tin về file ghi âm vào MediaStore.
2. **Phát lại:**
    - Sử dụng `MediaPlayer` để phát lại file ghi âm.
    - Có thể phát từ file hoặc từ một URI trong MediaStore.
3. **Hiển thị danh sách file ghi âm:**
    - Sử dụng `ContentResolver` để truy vấn MediaStore và lấy danh sách các file âm thanh.
    - Hiển thị danh sách này lên giao diện người dùng (ví dụ: ListView).

BÀI TẬP 7: Ứng dụng chơi video đơn giản

Mục tiêu:

- Sử dụng `VideoView` và `MediaController` để phát video.
- Cho phép người dùng chọn video từ MediaStore hoặc từ một URL.

Mô tả:

Ứng dụng cho phép người dùng xem video từ các nguồn khác nhau trên thiết bị hoặc từ Internet.

Các bước thực hiện:

1. **Thêm VideoView và MediaController vào layout.**
2. **Chọn video:**
    - Cho phép người dùng chọn video từ MediaStore bằng cách sử dụng `Intent.ACTION_PICK` và `MediaStore.Video.Media.EXTERNAL_CONTENT_URI`.
    - Hoặc cho phép người dùng nhập URL của video.
3. **Phát video:**
    - Sử dụng `VideoView.setVideoURI()` để thiết lập nguồn video.
    - Sử dụng `MediaController` để cung cấp các điều khiển phát lại video (play, pause, stop,...).
    - `VideoView.start()` để bắt đầu phát video.

Lưu ý:

- **Quyền (Permissions):** Đừng quên khai báo các quyền cần thiết trong `AndroidManifest.xml`, chẳng hạn như `android.permission.RECORD_AUDIO`, `android.permission.READ_EXTERNAL_STORAGE`, `android.permission.WRITE_EXTERNAL_STORAGE`, và `android.permission.INTERNET`.
- **Xử lý lỗi:** Cần xử lý các trường hợp lỗi có thể xảy ra, chẳng hạn như không tìm thấy file, lỗi khi ghi âm, lỗi khi phát lại, v.v.
- **Vòng đời (Lifecycle):** Quản lý các tài nguyên Media một cách chính xác trong các phương thức lifecycle của Activity/Fragment để tránh rò rỉ bộ nhớ và các vấn đề khác. Ví dụ, cần `release()` MediaPlayer và MediaRecorder khi không còn sử dụng.
- **MediaStore:** Làm quen với cách truy vấn và sử dụng MediaStore để lấy thông tin về các file media trên thiết bị.

BÀI TẬP 8: Ứng dụng đo gia tốc

Mục tiêu:

- Sử dụng cảm biến gia tốc (TYPE_ACCELEROMETER) để đo gia tốc của thiết bị theo ba trục x, y, z.
- Hiển thị giá trị gia tốc lên TextView.
- Hiển thị một hình ảnh động (ví dụ: một quả bóng) di chuyển theo hướng gia tốc.

Mô tả:

Ứng dụng hiển thị các giá trị gia tốc đo được từ cảm biến gia tốc và mô phỏng sự di chuyển của một vật thể dựa trên các giá trị này.

Các bước thực hiện:

1. **Lấy SensorManager và Sensor:**
   - Lấy `SensorManager` từ hệ thống.
   - Sử dụng `SensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)` để lấy đối tượng `Sensor`.
2. **Đăng ký SensorEventListener:**
   - Đăng ký một `SensorEventListener` để lắng nghe các sự kiện từ cảm biến gia tốc.
   - Implement hai phương thức của `SensorEventListener`:
     - `onSensorChanged(SensorEvent event)`: Xử lý các sự kiện cảm biến, lấy giá trị gia tốc từ `event.values`.
     - `onAccuracyChanged(Sensor sensor, int accuracy)`: Xử lý khi độ chính xác của cảm biến thay đổi.
3. **Hiển thị giá trị gia tốc:**
   - Cập nhật các TextView để hiển thị giá trị gia tốc theo trục x, y, z.
4. **Mô phỏng chuyển động:**
   - Sử dụng một `ImageView` để hiển thị hình ảnh động.
   - Trong phương thức `onSensorChanged()`, tính toán sự thay đổi vị trí của hình ảnh dựa trên giá trị gia tốc.
   - Cập nhật vị trí của `ImageView`.

BÀI TẬP 9: Ứng dụng la bàn

Mục tiêu:

- Sử dụng cảm biến từ trường (TYPE_MAGNETIC_FIELD) và cảm biến gia tốc (TYPE_ACCELEROMETER) để xác định hướng bắc.
- Hiển thị hướng bắc trên một `ImageView` (ví dụ: kim la bàn).
- Hiển thị góc lệch so với hướng bắc.

Mô tả:

Ứng dụng hiển thị la bàn và hướng bắc dựa trên dữ liệu từ cảm biến từ trường và cảm biến gia tốc.

Các bước thực hiện:

1. **Lấy SensorManager và Sensor:**
   - Lấy `SensorManager` từ hệ thống.
   - Sử dụng `SensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)` để lấy đối tượng `Sensor` từ trường.
   - Sử dụng `SensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)` để lấy đối tượng `Sensor` gia tốc.
2. **Đăng ký SensorEventListener:**
   - Đăng ký một `SensorEventListener` để lắng nghe các sự kiện từ cảm biến từ trường và gia tốc.
   - Trong phương thức `onSensorChanged()`:
     - Lấy giá trị từ cả hai cảm biến.
     - Sử dụng các hàm `SensorManager.getRotationMatrix()` và `SensorManager.getOrientation()` để tính toán hướng bắc và góc lệch.
3. **Hiển thị la bàn:**
   - Sử dụng một `ImageView` để hiển thị hình ảnh la bàn.
   - Sử dụng phương thức `ImageView.setRotation()` để xoay hình ảnh la bàn theo hướng bắc.
4. **Hiển thị góc lệch:**
   - Hiển thị giá trị góc lệch so với hướng bắc lên một `TextView`.

Lưu ý:

- **Quyền (Permissions):** Khai báo các quyền cần thiết trong `AndroidManifest.xml`, bao gồm quyền sử dụng cảm biến.
- **Độ chính xác:** Độ chính xác của cảm biến có thể bị ảnh hưởng bởi nhiễu từ môi trường. Cần có các biện pháp lọc dữ liệu hoặc hiệu chỉnh để cải thiện độ chính xác.
- **Tiết kiệm pin:** Hủy đăng ký `SensorEventListener` khi không sử dụng cảm biến để tiết kiệm pin.
- **Kiểm tra cảm biến:** Kiểm tra xem thiết bị có hỗ trợ các cảm biến cần thiết hay không trước khi sử dụng.

BÀI TẬP 10: ỨNG DỤNG CLIENT-SERVER ĐƠN GIẢN SỬ DỤNG TCP

Mục tiêu:

- Xây dựng một ứng dụng Android (Client) có thể gửi và nhận dữ liệu từ một ứng dụng Server (có thể chạy trên PC hoặc thiết bị Android khác) sử dụng giao thức TCP.
- Ứng dụng Client cho phép người dùng nhập tin nhắn và gửi đến Server.
- Ứng dụng Server nhận tin nhắn và hiển thị chúng.

Mô tả:

Ứng dụng này minh họa giao tiếp cơ bản giữa Client và Server sử dụng TCP, tập trung vào việc thiết lập kết nối, gửi và nhận dữ liệu.

Các bước thực hiện:

Phía Server:

1. **Tạo ServerSocket:** Sử dụng `ServerSocket` để lắng nghe kết nối đến trên một cổng cụ thể.
2. **Chấp nhận kết nối:** Sử dụng `ServerSocket.accept()` để chấp nhận kết nối từ Client.
3. **Tạo luồng (Thread):** Tạo một luồng mới để xử lý giao tiếp với mỗi Client.
4. **Giao tiếp:** Sử dụng `InputStream` và `OutputStream` của `Socket` để nhận và gửi dữ liệu.
5. **Đóng kết nối:** Đóng `Socket` và `ServerSocket` khi hoàn tất giao tiếp.

Phía Client (Android):

1. **Tạo Socket:** Sử dụng `Socket` để kết nối đến Server trên địa chỉ IP và cổng cụ thể.
2. **Giao tiếp:** Sử dụng `InputStream` và `OutputStream` của `Socket` để gửi và nhận dữ liệu.
3. **Gửi dữ liệu:** Lấy dữ liệu từ người dùng (ví dụ: một EditText) và gửi đến Server.
4. **Nhận dữ liệu:** Nhận phản hồi từ Server và hiển thị cho người dùng.
5. **Đóng kết nối:** Đóng `Socket` khi hoàn tất giao tiếp.

BÀI TẬP 11: Ứng dụng gửi và nhận tin nhắn sử dụng UDP

Mục tiêu:

- Xây dựng một ứng dụng Android có thể gửi và nhận tin nhắn sử dụng giao thức UDP.
- Ứng dụng cho phép người dùng gửi tin nhắn đến một thiết bị khác trên mạng.
- Ứng dụng có thể nhận tin nhắn từ các thiết bị khác.

Mô tả:

Ứng dụng này minh họa giao tiếp sử dụng UDP, tập trung vào việc gửi và nhận các gói tin độc lập.

Các bước thực hiện:

Gửi tin nhắn:

1. **Lấy dữ liệu:** Lấy dữ liệu từ người dùng (ví dụ: một EditText).
2. **Tạo DatagramPacket:** Tạo một `DatagramPacket` chứa dữ liệu cần gửi, địa chỉ IP và cổng của người nhận.
3. **Tạo DatagramSocket:** Tạo một `DatagramSocket` để gửi gói tin.
4. **Gửi gói tin:** Sử dụng `DatagramSocket.send()` để gửi `DatagramPacket`.
5. **Đóng DatagramSocket:** Đóng `DatagramSocket` sau khi gửi.

Nhận tin nhắn:

1. **Tạo DatagramSocket:** Tạo một `DatagramSocket` và lắng nghe trên một cổng cụ thể.
2. **Tạo DatagramPacket:** Tạo một `DatagramPacket` để chứa dữ liệu nhận được.
3. **Nhận gói tin:** Sử dụng `DatagramSocket.receive()` để nhận `DatagramPacket`.
4. **Xử lý dữ liệu:** Trích xuất dữ liệu từ `DatagramPacket` và hiển thị.
5. **Đóng DatagramSocket:** Đóng `DatagramSocket` khi không còn cần nhận tin nhắn.

Lưu ý:

- **Quyền (Permissions):** Đảm bảo khai báo các quyền cần thiết trong `AndroidManifest.xml`, bao gồm `android.permission.INTERNET`.
- **Luồng (Threads):** Thực hiện các hoạt động mạng trong các luồng riêng để tránh làm treo UI thread.
- **Xử lý lỗi:** Xử lý các trường hợp lỗi có thể xảy ra, chẳng hạn như kết nối không thành công, mất kết nối, v.v.
- **Giao thức:** Xác định rõ giao thức giao tiếp giữa Client và Server (ví dụ: định dạng tin nhắn, các lệnh).