private void saveUserCredentials(String username, String password) { }

SharedPreferences 클래스는 안드로이드에서 간단한 데이터를 저장하고 불러오는 데 사용되는 인터페이스입니다. 이를 통해 애플리케이션 내에서 설정값이나 작은 규모의 데이터를 키-값(pair) 형태로 영구적으로 저장할 수 있습니다.

savedUserCredentials: String타입의 username과 password라는 두개의 매개변수를 받아들이는 private 메소드 입니다.

이메소드는 사용자가 입력한 아이디와 비밀번호를 저장하기 위한 목적으로 작성되었습니다.

SharedPreferences prefs = getSharedPreferences(“UserPrefs”, MODE\_PRIVATE);

getSharedPreferences("UserPrefs", MODE\_PRIVATE)를 호출하여 이름이 "UserPrefs"인 SharedPreferences 파일을 엽니다. 이 파일은 애플리케이션의 설정이나 데이터를 저장하기 위해 사용됩니다.

MODE\_PRIVATE은 이 파일이 오직 이 애플리케이션에서만 접근 가능하도록 합니다.

prefs는 이 SharedPreferences 객체를 가리키는 변수입니다.

SharedPreferencse.Editor editor = prefs.edit();

SharedPreferences를 수정하기 위해 edit() 메서드를 호출하여 SharedPreferences.Editor 객체를 생성합니다.

editor 변수는 SharedPreferences.Editor 객체를 가리키며, 이 객체를 통해 SharedPreferences에 데이터를 저장하거나 수정할 수 있습니다.

editor.putString("username", username);

editor.putString("password", password);

putString(String key, String value) 메서드는 주어진 키("username", "password")와 그에 대응하는 값(username, password)을 SharedPreferences에 저장합니다.

"username"이라는 키에는 사용자가 입력한 username이 저장되고, "password"라는 키에는 사용자가 입력한 password가 저장됩니다.

editor.apply();

apply() 메서드는 변경된 내용을 비동기적으로 SharedPreferences에 저장합니다. apply()는 commit()과 유사하지만, 차이점은 apply()가 비동기적으로 처리되어 UI 스레드가 차단되지 않는다는 점입니다.

이 호출을 통해 실제로 username과 password 값이 SharedPreferences에 저장됩니다.

이 saveUserCredentials 메서드는 사용자가 입력한 username과 password를 안드로이드의 SharedPreferences라는 영구 저장소에 저장합니다. 저장된 데이터는 애플리케이션이 다시 실행되더라도 유지되며, 다음에 이 저장된 자격 증명을 불러와 사용자의 로그인 상태를 확인하거나, 다른 필요한 작업을 수행할 수 있습니다.

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

private boolean checkLogin(String username, String password) { }

checkLogin은 username과 password라는 두 개의 문자열을 매개변수로 받아들이는 private 메서드입니다.

이 메서드는 반환값으로 boolean을 리턴합니다. 즉, 로그인 정보가 일치하면 true, 일치하지 않으면 false를 반환합니다.

SharedPreferences prefs = getSharedPreferences("UserPrefs", MODE\_PRIVATE);

getSharedPreferences("UserPrefs", MODE\_PRIVATE)를 호출하여 "UserPrefs"라는 이름의 SharedPreferences 파일을 엽니다. 이 파일은 사용자의 자격 증명(사용자 이름과 비밀번호)을 저장하는 데 사용되었습니다.

MODE\_PRIVATE은 이 파일이 이 앱 내에서만 접근 가능하게 합니다.

prefs는 이 SharedPreferences 객체를 가리킵니다.

String savedUsername = prefs.getString("username", "");

String savedPassword = prefs.getString("password", "");

prefs.getString("username", "")는 "username" 키로 저장된 값을 불러옵니다. 만약 해당 키에 저장된 값이 없다면, 기본값으로 빈 문자열 ""을 반환합니다.

prefs.getString("password", "")는 "password" 키로 저장된 값을 불러옵니다. 마찬가지로 해당 키에 저장된 값이 없다면, 기본값으로 빈 문자열 ""을 반환합니다.

savedUsername과 savedPassword는 각각 저장된 사용자 이름과 비밀번호를 나타냅니다.

return username.equals(savedUsername) && password.equals(savedPassword);

입력된 username과 저장된 savedUsername을 비교하고, 입력된 password와 저장된 savedPassword를 비교합니다.

두 조건이 모두 참(true)일 경우에만 true를 반환하고, 그렇지 않으면 false를 반환합니다.

즉, 사용자가 입력한 자격 증명이 저장된 값과 정확히 일치하면 로그인에 성공하게 되고, 이때 true를 반환합니다.

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

checkAnswer()

answered 라는 변수에 true를 할당하여 사용자가 현재 질문에 답변했음을 표시한다.

RadioButton rbSelected = findViewById(radioGroup.getCheckedRadioButtonId());

Rbselected는 사용자가 선택한 라디오 버튼을 참조한다.

int selectedAnswerNo = radioGroup.indexOfChild(rbSelected) + 1;

radioGroup.indexofChild(rbselected)는 선택된 라디오 버튼이 radioGroup 내에서 몇 번째 위치에 있는지를 반환한다. 이 인덱스는 0부터 시작하기 때문에 이를 실제 답변 번호와 맞추기 위해 +1을 더한다.

Question currentQuestion = questionList.get(currentQuestionIndex - 1); // 현재 문제를 가져옴

questionList에서 현재 질문을 가져온다. currentQusetionIndex는 현재 질문의 인덱스를 나타내며 인덱스는 1부터 시작한다고 가정하므로 -1을 해서 리스트에서 올바른 질문을 가져온다.

currentQuestion은 현재 사용자가 답변하는 질문을 나타낸다.

if (selectedAnswerNo == currentQuestion.getCorrectAnswerNo()) {

score++;

scoreTextView.setText("Score: " + score);

}

사용자가 선택한 답변 번호{selectedAnswerNo}가 현재 질문의 정답번호(currentQuestion.getCorrectAnswerNo()와 일치하는지 확인한다.

만약 일치하면 사용자의 점수를 증가시킨다.(score++)

증가된 점수를 scoreTextView에 업데이트하여 사용자에게 현재 점수를 표시한다.

else {

showCorrectAnswer(currentQuestion);

} 사용자가 선택한 답변이 정답이 아니라면 showCorrectAnswer(currentQuestion)을 호출하여 정답을 보여준다.

if (currentQuestionIndex < questionList.size()) {

submitButton.setText("Next");

} else {

submitButton.setText("Finish");

}.

현재 질문의 인덱스(currentQuestionIndex)가 전체 질문 리스트(questionList)의 크기보다 작으면(즉, 아직 다음 질문이 남아 있으면) submitButton의 텍스트를 "Next"로 설정합니다.

모든 질문을 다 답변한 경우(즉, 마지막 질문에 도달했으면) 버튼의 텍스트를 "Finish"로 설정하여 퀴즈가 종료될 것임을 알립니다.

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

private void showCorrectAnswer(Question currentQuestion) { }

showCorrectAnswer는 currentQuestion 이라는 Question 객체를 매개변수로 받아 해당질문의 정답을 화면에 표시하는 메소드이다.

모든 선택지를 빨간색으로 표시하여 오답을 시각적으로 강조한 후, 정답에 해당하는 선택지의 텍스트 색상을 초록색으로 바꿔 정답을 표시합니다.

이렇게 함으로써 사용자는 자신의 선택이 틀렸다는 것을 인식하고, 정답이 무엇인지 명확히 알 수 있게 됩니다.

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

radioGroup.clearCheck();

사용자가 이전 질문에서 선택한라디오 버튼의 선택을 해제합니다. 즉 라디오 그룹에서 모든 선택이 초기화 됩니다.

네 개의 라디오 버튼(rb1, rb2, rb3, rb4)의 텍스트 색상을 모두 검은색(Color.BLACK)으로 설정합니다.

이전 질문에서 오답과 정답이 각각 빨간색과 초록색으로 표시되었을 수 있는데, 이를 다음 질문에서 초기화하여 기본 상태(검은색)로 되돌립니다.

if (currentQuestionIndex < questionList.size()) {

현재 질문의 인덱스가 전체 질문 리스트의 크기보다 작은지 확인합니다. 즉 아직 다음 질문이 남아있는지 확인하는 조건입니다.

Question currentQuestion = questionList.get(currentQuestionIndex);

questionList에서 currentQuestionIndex에 해당하는 질문을 가져와 currentQuestion에 할당합니다.

이 currentQuestion 객체는 현재 표시될 질문을 나타냅니다.

questionNumberTextView.setText("Question " + (currentQuestionIndex + 1));

현재 질문 번호를 표시합니다. currentQuestionIndex는 0부터 시작하므로, 실제 질문 번호는 currentQuestionIndex + 1입니다.

이를 questionNumberTextView에 설정하여 사용자에게 현재 몇 번째 질문인지 알려줍니다.

questionTextView.setText(currentQuestion.getQuestion());

rb1.setText(currentQuestion.getOption1());

rb2.setText(currentQuestion.getOption2());

rb3.setText(currentQuestion.getOption3());

rb4.setText(currentQuestion.getOption4());

urrentQuestion 객체에서 질문과 선택지를 가져와 각각의 텍스트 뷰와 라디오 버튼에 설정합니다.

questionTextView에는 현재 질문의 텍스트를 설정합니다.

rb1, rb2, rb3, rb4에는 각각 네 가지 선택지를 설정합니다.

answered = false;

answered 변수를 false로 설정하여, 현재 질문이 아직 답변되지 않았음을 나타냅니다.

이는 사용자가 아직 답을 제출하지 않았음을 의미합니다.

submitButton.setText("Submit");

submitButton의 텍스트를 "Submit"으로 설정합니다. 사용자가 현재 질문에 답을 제출할 수 있도록 버튼의 역할을 명확히 합니다.

currentQuestionIndex++;

currentQuestionIndex를 증가시켜, 다음 질문으로 이동할 준비를 합니다.

만약 currentQuestionIndex가 questionList.size()와 같거나 크다면(즉, 더 이상 남아 있는 질문이 없으면), 퀴즈가 종료되었음을 사용자에게 알립니다.

Toast.makeText(this, "Quiz Finished!", Toast.LENGTH\_SHORT).show();는 작은 팝업 메시지를 화면에 표시하여 퀴즈가 끝났음을 알려줍니다.

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Question 클래스는 각 질문과 관련된 모든 데이터를 저장하고, 이 데이터를 다른 메서드에서 쉽게 사용할 수 있도록 합니다.

showNextQuestion 메서드는 Question 객체에서 질문과 선택지를 가져와 화면에 표시합니다.

checkAnswer 메서드는 Question 객체에서 정답을 확인하고, 정답 여부에 따라 점수를 계산하거나 정답을 시각적으로 표시하는 showCorrectAnswer 메서드를 호출합니다.

Question 클래스는 퀴즈 로직의 중심 역할을 하며, 퀴즈가 정상적으로 진행되도록 돕습니다.

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////public class WordDatabaseHelper extends SQLiteOpenHelper { }

WordDatabaseHelper 클래스는 SQLiteOpenHelper를 상속받아 데이터베이스를 관리하는 헬퍼 클래스입니다.

private static final String DATABASE\_NAME = "words.db";

private static final int DATABASE\_VERSION = 1;

DATABASE\_NAME은 데이터베이스 파일의 이름으로, "words.db"라고 설정되어 있습니다.

DATABASE\_VERSION은 데이터베이스의 버전을 나타내며, 현재 1로 설정되어 있습니다. 데이터베이스 스키마가 변경되면 이 값을 증가시켜야 합니다.

테이블 및 컬럼 이름 선언:

public static final String TABLE\_WORDS = "words";

public static final String COLUMN\_ID = "\_id";

public static final String COLUMN\_WORD = "word";

public static final String COLUMN\_DESCRIPTION = "description";

ABLE\_WORDS: 테이블의 이름을 정의하며, "words"라는 테이블을 생성합니다.

COLUMN\_ID: 각 레코드의 고유 ID를 저장하는 컬럼으로, 주로 기본 키 역할을 합니다.

COLUMN\_WORD: 단어를 저장하는 컬럼입니다.

COLUMN\_DESCRIPTION: 단어의 설명을 저장하는 컬럼입니다.

테이블 생성 SQL 명령문:

private static final String TABLE\_CREATE =

"CREATE TABLE " + TABLE\_WORDS + " (" +

COLUMN\_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +

COLUMN\_WORD + " TEXT, " +

COLUMN\_DESCRIPTION + " TEXT" +

");";

TABLE\_CREATE는 SQL 명령문으로, words 테이블을 생성합니다.

이 명령문은 COLUMN\_ID를 기본 키로 설정하며, 자동으로 증가하도록(AUTOINCREMENT) 합니다.

COLUMN\_WORD와 COLUMN\_DESCRIPTION은 텍스트 타입으로 설정됩니다.

public WordDatabaseHelper(Context context) {

super(context, DATABASE\_NAME, null, DATABASE\_VERSION);

}

이 생성자는 SQLiteOpenHelper의 생성자를 호출하여 데이터베이스 헬퍼를 초기화합니다.

context는 애플리케이션의 컨텍스트를 전달합니다.

DATABASE\_NAME과 DATABASE\_VERSION은 앞서 정의한 값들로, 데이터베이스의 이름과 버전을 설정합니다.

@Override

public void onCreate(SQLiteDatabase db) {

db.execSQL(TABLE\_CREATE);

}

onCreate 메서드는 데이터베이스가 처음 생성될 때 호출됩니다.

이 메서드는 TABLE\_CREATE 명령문을 실행하여 데이터베이스에 words 테이블을 생성합니다.

dbHelper = new WordDatabaseHelper(this);

WordDatabaseHelper의 인스턴스를 생성하여 dbHelper 변수에 할당합니다.

이 객체는 SQLite 데이터베이스와의 상호작용(읽기, 쓰기 등)을 관리합니다.

this는 현재의 Context(여기서는 Activity)를 의미하며, 데이터베이스 관리에 필요합니다.

private void addWordToDatabase() {

String word = wordEditText.getText().toString().trim();

String description = descriptionEditText.getText().toString().trim();

addWordToDatabase 메서드는 사용자가 입력한 단어와 설명을 데이터베이스에 추가하는 기능을 합니다.

wordEditText와 descriptionEditText에서 사용자가 입력한 텍스트를 가져와 각각 word와 description 변수에 저장합니다.

.trim() 메서드를 사용하여 입력 문자열의 앞뒤 공백을 제거합니다.

if (word.isEmpty() || description.isEmpty()) {

Toast.makeText(this, "Please fill out both fields", Toast.LENGTH\_SHORT).show();

return;

}

사용자가 단어나 설명 중 하나라도 입력하지 않았을 경우, 입력이 부족하다는 메시지를 토스트로 표시하고 메서드를 종료합니다.

isEmpty() 메서드는 문자열이 비어 있는지 확인합니다.

SQLiteDatabase db = dbHelper.getWritableDatabase();

dbHelper 객체를 통해 데이터베이스의 쓰기 가능한 인스턴스(SQLiteDatabase)를 가져옵니다.

이 인스턴스를 사용해 데이터베이스에 데이터를 추가하거나 수정할 수 있습니다.

ContentValues values = new ContentValues();

values.put(WordDatabaseHelper.COLUMN\_WORD, word);

values.put(WordDatabaseHelper.COLUMN\_DESCRIPTION, description);

ContentValues 객체를 생성하여 데이터베이스에 삽입할 데이터를 준비합니다.

put 메서드를 사용해 COLUMN\_WORD와 COLUMN\_DESCRIPTION에 각각 단어와 설명을 넣습니다.

ContentValues는 데이터베이스에 데이터를 삽입할 때 사용되는 키-값 쌍의 집합을 제공합니다.

long newRowId = db.insert(WordDatabaseHelper.TABLE\_WORDS, null, values);

insert 메서드를 사용해 values에 담긴 데이터를 words 테이블에 삽입합니다.

newRowId는 삽입된 행의 ID를 반환합니다. 만약 삽입이 실패하면 -1이 반환됩니다.

insert 메서드를 사용해 values에 담긴 데이터를 words 테이블에 삽입합니다.

newRowId는 삽입된 행의 ID를 반환합니다. 만약 삽입이 실패하면 -1이 반환됩니다.

newRowId가 -1이면 삽입에 실패한 것이므로, "Error saving word"라는 메시지를 표시합니다.

삽입이 성공하면 "Word added"라는 메시지를 표시하고, wordEditText와 descriptionEditText를 비웁니다(즉, 입력 필드를 초기화합니다).

그 후 MainActivity4로 화면을 전환하는 인텐트를 생성하고, startActivity(intent)로 전환을 수행합니다.

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

이 코드는 SQLite 데이터베이스에서 단어와 설명을 조회하여 `ListView`에 표시하는 기능을 구현한 것입니다. `SimpleCursorAdapter`를 사용하여 `Cursor` 데이터를 `ListView`에 연결합니다.

private WordDatabaseHelper dbHelper;

private SimpleCursorAdapter adapter;

- `dbHelper`는 `WordDatabaseHelper`의 인스턴스를 나타내며, 데이터베이스에 접근하는 역할을 합니다.

- `adapter`는 `SimpleCursorAdapter`의 인스턴스로, 데이터베이스에서 가져온 데이터를 `ListView`에 연결하여 화면에 표시하는 데 사용됩니다.

dbHelper = new WordDatabaseHelper(this);

- `WordDatabaseHelper`의 인스턴스를 생성하여 `dbHelper` 변수에 할당합니다.

- `this`는 현재의 `Context`(예: `Activity`)를 의미하며, 데이터베이스 관리에 필요합니다.

displayWords(listView);

- `displayWords` 메서드를 호출하여 데이터베이스에서 단어와 설명을 조회하고, 결과를 `listView`에 표시합니다.

### `displayWords` 메서드:

private void displayWords(ListView listView) {

SQLiteDatabase db = dbHelper.getReadableDatabase();

- `displayWords` 메서드는 `ListView`에 단어 목록을 표시합니다.

- `dbHelper.getReadableDatabase()`를 호출하여 데이터베이스의 읽기 가능한 인스턴스를 얻습니다.

String[] projection = { WordDatabaseHelper.COLUMN\_ID,

WordDatabaseHelper.COLUMN\_WORD,

WordDatabaseHelper.COLUMN\_DESCRIPTION };

- `projection` 배열은 쿼리에서 선택할 컬럼을 정의합니다.

- `COLUMN\_ID`, `COLUMN\_WORD`, `COLUMN\_DESCRIPTION`은 각각 ID, 단어, 설명 컬럼을 의미합니다.

Cursor cursor = db.query( WordDatabaseHelper.TABLE\_WORDS,

projection,

null,

null,

null,

null,

null

);

- `db.query` 메서드를 호출하여 `TABLE\_WORDS` 테이블에서 데이터를 조회합니다.

- `projection`에 지정된 컬럼들만 선택됩니다.

- 나머지 매개변수(`selection`, `selectionArgs`, `groupBy`, `having`, `orderBy`)는 `null`로 설정되어 있으며, 이는 모든 행을 조회하고 필터링이나 정렬을 하지 않음을 의미합니다.

- 결과는 `Cursor` 객체로 반환됩니다. `Cursor`는 데이터베이스 쿼리 결과를 행 단위로 순회할 수 있는 객체입니다.

String[] fromColumns = { WordDatabaseHelper.COLUMN\_WORD,

WordDatabaseHelper.COLUMN\_DESCRIPTION };

int[] toViews = {R.id.wordTextView, R.id.descriptionTextView };

- `fromColumns` 배열은 `Cursor`에서 가져올 데이터의 컬럼 이름을 정의합니다.

- `toViews` 배열은 레이아웃에서 `TextView`의 ID를 정의합니다. 이 `TextView`는 각 항목의 단어와 설명을 표시합니다.

adapter = new SimpleCursorAdapter(

this,

R.layout.list\_item,

cursor,

fromColumns,

toViews,

0

);

- `SimpleCursorAdapter`를 생성하여 `adapter` 변수에 할당합니다.

- `SimpleCursorAdapter`는 `Cursor` 데이터를 `ListView`의 항목으로 변환하는 역할을 합니다.

- 생성자 매개변수:

- `this`: 현재 `Context`를 의미합니다.

- `R.layout.list\_item`: `ListView` 항목의 레이아웃 리소스입니다.

- `cursor`: 데이터를 제공하는 `Cursor` 객체입니다.

- `fromColumns`: `Cursor`에서 데이터를 가져올 컬럼입니다.

- `toViews`: `list\_item` 레이아웃에서 데이터가 표시될 `TextView`의 ID입니다.

- `0`: 플래그를 설정할 수 있는 매개변수로, 기본값 `0`은 특별한 동작이 없음을 의미합니다.

listView.setAdapter(adapter);

- `listView.setAdapter(adapter)`를 호출하여 `ListView`에 `SimpleCursorAdapter`를 설정합니다.

- `ListView`는 이제 데이터베이스에서 가져온 데이터를 `ListView` 항목으로 표시하게 됩니다.

### 요약:

- 이 코드의 주요 기능은 SQLite 데이터베이스에서 단어와 설명을 조회하고, 이를 `ListView`에 표시하는 것입니다.

- `SimpleCursorAdapter`를 사용하여 `Cursor` 데이터와 `ListView` 항목 레이아웃을 연결합니다.

- 데이터베이스에서 가져온 데이터는 `ListView`의 각 항목으로 표시되며, 각 항목은 단어와 설명을 포함합니다.

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

scrambledWord = shuffleWord(originalWord);

tvScrambledWord.setText(scrambledWord);

shuffleWord(originalWord) 메서드를 호출하여 원래 단어(originalWord)의 문자들을 섞습니다.

scrambledWord에 섞인 단어를 저장하고, 이를 tvScrambledWord라는 TextView에 표시합니다.

btnSubmit.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

String userGuess = etUserInput.getText().toString().trim();

checkGuess(userGuess);

}

}); btnSubmit 버튼에 클릭 리스너를 설정합니다. 사용자가 버튼을 클릭하면, etUserInput라는 EditText에서 사용자가 입력한 텍스트를 가져옵니다.

사용자의 입력 텍스트(userGuess)를 checkGuess(userGuess) 메서드에 전달하여 답안을 확인합니다.

showAnswerButton.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

// 정답을 텍스트뷰에 표시

tvScrambledWord.setText("정답: " + originalWord);

tvScrambledWord.setVisibility(View.VISIBLE);

}

}); showAnswerButton 버튼에 클릭 리스너를 설정합니다. 버튼이 클릭되면, tvScrambledWord라는 TextView에 정답(originalWord)을 "정답: "이라는 텍스트와 함께 표시합니다.

tvScrambledWord의 가시성을 설정하여 정답이 표시되도록 합니다.

private String shuffleWord(String word) {

List<Character> letters = new ArrayList<>();

for (char c : word.toCharArray()) {

letters.add(c);

}

Collections.shuffle(letters);

StringBuilder shuffledWord = new StringBuilder();

for (char c : letters) {

shuffledWord.append(c);

}

return shuffledWord.toString();

}

shuffleWord 메서드는 주어진 단어(word)의 각 문자들을 List에 추가합니다.

Collections.shuffle(letters)를 호출하여 문자를 무작위로 섞습니다.

StringBuilder를 사용하여 섞인 문자를 다시 문자열로 조합하여 반환합니다.

private void checkGuess(String guess) {

if (guess.equalsIgnoreCase(originalWord)) {

tvResult.setText("Correct! 정답은 " + originalWord);

} else {

tvResult.setText("Incorrect, try again.");

}

}

checkGuess 메서드는 사용자가 입력한 추측(guess)이 원래 단어(originalWord)와 일치하는지 확인합니다.

사용자의 입력이 맞으면 "Correct! 정답은 [단어]"라는 메시지를 tvResult라는 TextView에 표시합니다.

입력이 틀리면 "Incorrect, try again."이라는 메시지를 표시합니다.