RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Department of Master of Computer Applications (MCA)

Mobile Application Development (MCA221IA)

Hand Notes

Unit – UNIT 1	Topic: INTENT
Editor: 1RV24MC113 – VARUN SRIVASTAVA	

List of Questions

2 Marks Questions

Q1. Define Android Intent and explain its primary purpose in Android application development.

Q2. Differentiate between Explicit and Implicit Intents with one example each.

Q3. What is an Intent Filter? How does it help in handling implicit intents?

Q4. List any four types of data that can be passed through Intent extras.

Q5. What is the role of startActivity() method in Intent handling?

Q6. Explain the concept of Intent resolution in Android.

Q7. What are the key components required to create an explicit intent?

Q8. How do you pass data from one activity to another using Intent extras?

4 Marks Questions

Q1. Explain the structure of an Android Intent with its key components. Provide a code example demonstrating how to create and use an explicit intent to navigate between activities.

Q2. Compare and contrast Explicit and Implicit Intents. Write code snippets showing how to implement both types of intents for opening a new activity and sharing text content respectively.

Q3. Describe the process of Intent resolution in Android. How does the Android system determine which component should handle an implicit intent? Include the role of Intent filters in your explanation.

Q4. Explain how to pass different types of data (String, Integer, Boolean, and custom objects) between activities using Intent extras. Provide code examples for both sending and receiving data.

Q5. What are Intent filters and how are they declared in the Android manifest file? Write an example showing how to register an activity to handle custom actions through intent filters.

Q6. Describe the lifecycle of an Intent from creation to execution. Include the steps involved when using startActivity() and startActivityForResult() methods.

5 Marks Questions

Q1. Elaborate on the different types of Android Intents and their use cases. Provide practical examples with code snippets for:

- Starting a new activity
- Sending an email
- Making a phone call
- Opening a web page

Q2. Explain the concept of Intent flags and their significance in controlling activity behavior. Describe at least three commonly used flags with examples:

- FLAG_ACTIVITY_NEW_TASK
- FLAG_ACTIVITY_CLEAR_TOP
- FLAG_ACTIVITY_SINGLE_TOP

Q3. Describe the process of creating and handling custom actions using Intents. Include:

- Defining custom action strings
- Registering intent filters in manifest
- Creating and sending custom intents
- Handling received intents in the target activity

Q4. Discuss the security considerations when working with Intents in Android development. Cover:

- Intent spoofing and data leakage
- Best practices for validating incoming intents
- Using explicit intents for sensitive operations
- Proper handling of intent extras

Q5. Explain how to implement communication between activities using startActivityForResult() and onActivityResult(). Provide a complete code example showing data exchange between two activities including result codes and data validation.

6 Marks Questions

Q1. Design a comprehensive solution for inter-activity communication in an Android application. Your answer should include:

- Implementation of explicit intents for navigation
- Passing complex data structures between activities
- Handling activity results and return data
- Error handling and validation
- Complete code examples with explanation

Q2. Analyze the role of Intent filters in Android application architecture. Discuss:

- Different types of intent filter elements (action, category, data)
- Priority and precedence handling
- Creating a custom content provider accessible through intents

- Best practices for intent filter design
- Code examples demonstrating various filter configurations

Q3. Evaluate the performance implications of different Intent usage patterns in Android applications. Cover:

- Memory management with large data transfers
- Alternative approaches for sharing data (Application class, SharedPreferences, databases)
- Optimization techniques for intent handling
- When to use different communication methods
- Practical implementation guidelines

Q4. Develop a detailed explanation of how to integrate Intent handling in a Flutter Android application. Include:

- Platform channel implementation for native Android intents
- Handling deep links and app shortcuts
- Managing intent data in Flutter widgets
- Code examples for bidirectional communication
- Error handling and platform-specific considerations

10 Marks Questions

Q1. Comprehensive Intent Management System Design

Design and implement a complete intent management system for a multi-activity Android application. Your solution should address:

a) **Architecture Design (3 marks)**

- Intent routing strategy
- Data flow management
- Activity lifecycle integration

b) **Implementation Components (4 marks)**

- Custom intent builder classes
- Data serialization and deserialization utilities
- Intent validation and security measures
- Error handling mechanisms

c) **Advanced Features (3 marks)**

- Deep linking implementation
- Intent-based navigation patterns
- Performance optimization techniques

Provide complete code examples, UML diagrams, and explain how your system handles various edge cases and security concerns.

Q2. Cross-Platform Intent Integration: Android and Flutter

Develop a comprehensive solution for handling Android Intents in a Flutter application. Your answer should cover:

a) **Platform Channel Architecture (3 marks)**

- Method channel implementation
- Event channel for intent listening
- Data type mapping between Dart and Java/Kotlin

b) **Intent Handling Implementation (4 marks)**

- Native Android intent processing
- Flutter widget integration
- State management for intent data
- Deep link handling in Flutter

c) **Advanced Integration Scenarios (3 marks)**

- File sharing between apps
- Camera and gallery integration
- Custom URL scheme handling
- Background intent processing

Include complete code examples for both Android (Java/Kotlin) and Flutter (Dart), architectural diagrams, and discuss testing strategies for cross-platform intent handling.

Q3. Progressive Web App and Native Intent Integration

Analyze and implement intent-like functionality in Progressive Web Apps while maintaining compatibility with native Android intents. Address:

a) **Web App Manifest and Intent Handling (3 marks)**

- PWA configuration for intent handling
- Web Share API implementation
- Custom protocol handling

b) **Native-Web Bridge Implementation (4 marks)**

- JavaScript to native communication
- WebView intent interception
- Data synchronization between web and native components

c) **Security and Performance Considerations (3 marks)**

- Cross-origin security policies
- Performance optimization for web-native communication
- Offline intent handling strategies

Provide working code examples, performance benchmarks, and security analysis for the proposed solution.

Q4. Enterprise-Level Intent Security and Management

Design a robust intent security framework for enterprise Android applications. Your comprehensive solution should include:

a) **Security Architecture (3 marks)**

- Intent validation and sanitization frameworks
- Permission-based intent routing
- Audit logging and monitoring systems

b) **Implementation Framework (4 marks)**

- Custom intent managers with security policies
- Encrypted data transfer mechanisms
- Role-based access control for intents
- Integration with enterprise security protocols

c) **Advanced Security Features (3 marks)**

- Intent forensics and debugging tools
- Runtime security policy enforcement
- Integration with mobile device management (MDM)
- Compliance with enterprise security standards

---

2 Marks Questions - Answers

A1. Define Android Intent and explain its primary purpose

An Android Intent is a messaging object that facilitates communication between different application components (activities, services, broadcast receivers). Its primary purpose is to request an action from another app component, enabling loose coupling between components and supporting both intra-app and inter-app communication.

A2. Differentiate between Explicit and Implicit Intents

Explicit Intent: Specifies the exact component to be invoked by providing the class name. Example: Intent intent = new Intent(this, SecondActivity.class);

Implicit Intent: Specifies the action to be performed without naming a specific component. Example: Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com"));

A3. What is an Intent Filter?

An Intent Filter is a declaration in the manifest file that specifies the types of intents an application component can handle. It helps the Android system determine which components can respond to implicit intents by matching actions, categories, and data specifications.

A4. Four types of data that can be passed through Intent extras

1. String data using putExtra(String key, String value)
2. Integer data using putExtra(String key, int value)
3. Boolean data using putExtra(String key, boolean value)
4. Serializable objects using putExtra(String key, Serializable value)

A5. Role of startActivity() method

The startActivity() method is used to launch a new activity using an Intent. It tells the Android system to start the activity specified in the Intent object, handling the transition from the current activity to the target activity.

A6. Intent resolution in Android

Intent resolution is the process by which the Android system determines which component should handle an implicit intent. The system compares the intent's action, category, and data against the intent filters declared by various components to find the best match.

A7. Key components required for explicit intent

1. Context (current activity or application context)
2. Target component class (destination activity class)
3. Optional: Extra data to be passed
4. Optional: Flags to control activity behavior

A8. Passing data using Intent extras

Use putExtra() method to add data: intent.putExtra("key", value); Retrieve data using getIntent().getStringExtra("key") or appropriate getter method in the receiving activity.

4 Marks Questions - Answers

A1. Structure of Android Intent with explicit intent example

An Android Intent consists of:

- **Action**: What to do (e.g., ACTION_VIEW, ACTION_SEND)
- **Data**: URI of data to operate on
- **Category**: Additional information about the action
- **Extras**: Key-value pairs for additional data
- **Component**: Specific component to handle the intent
- **Flags**: Control activity behavior

```
// Explicit Intent Example
Intent intent = new Intent(MainActivity.this, SecondActivity.class);
intent.putExtra("message", "Hello from MainActivity");
intent.putExtra("number", 42);
startActivity(intent);
```

A2. Compare Explicit and Implicit Intents

Explicit Intents:

- Specify exact component to invoke
- Used for internal app communication
- More secure and predictable
- Direct component targeting

Implicit Intents:

- Specify action without naming component

- Used for inter-app communication
- System resolves appropriate handler
- Flexible but less predictable

```
// Explicit Intent
Intent explicit = new Intent(this, TargetActivity.class);
startActivity(explicit);

// Implicit Intent
Intent implicit = new Intent(Intent.ACTION_SEND);
implicit.setType("text/plain");
implicit.putExtra(Intent.EXTRA_TEXT, "Share this text");
startActivity(Intent.createChooser(implicit, "Share via"));
```

A3. Intent resolution process

Intent resolution involves:

1. **Intent Analysis**: System examines intent's action, category, and data
2. **Filter Matching**: Compares against declared intent filters
3. **Component Selection**: Identifies matching components
4. **Priority Resolution**: Handles multiple matches based on priority
5. **User Choice**: Presents chooser dialog if multiple options exist

Intent filters in manifest:

```
<intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <category android:name="android.intent.category.DEFAULT" />
  <data android:scheme="http" />
</intent-filter>
```

A4. Passing different data types via Intent extras

```
// Sending Activity
Intent intent = new Intent(this, ReceivingActivity.class);
intent.putExtra("stringData", "Hello World");
intent.putExtra("intData", 100);
intent.putExtra("booleanData", true);

// For custom objects (must implement Serializable or Parcelable)
CustomObject obj = new CustomObject();
intent.putExtra("customObject", obj);
startActivity(intent);

// Receiving Activity
String stringValue = getIntent().getStringExtra("stringData");
int intValue = getIntent().getIntExtra("intData", 0);
boolean boolValue = getIntent().getBooleanExtra("booleanData", false);
CustomObject obj = (CustomObject) getIntent().getSerializableExtra("customObject");
```

A5. Intent filters declaration and usage

Intent filters are declared in AndroidManifest.xml:

```
<activity android:name=".CustomActivity">
    <intent-filter>
        <action android:name="com.example.CUSTOM_ACTION" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

Usage:

```
Intent customIntent = new Intent("com.example.CUSTOM_ACTION");
customIntent.setType("text/plain");
customIntent.putExtra("data", "Custom data");
startActivity(customIntent);
```

A6. Intent lifecycle with startActivity() and startActivityForResult()

startActivity() lifecycle:

1. Intent creation and configuration
2. System intent resolution
3. Target activity instantiation
4. Current activity paused
5. New activity started and resumed

startActivityForResult() lifecycle:

1. Intent creation with request code
2. Target activity processing
3. Result setting using setResult()
4. Activity finish and return
5. onActivityResult() callback in calling activity

```
// Starting for result
startActivityForResult(intent, REQUEST_CODE);

// Handling result
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
        String result = data.getStringExtra("result");
    }
}
```

5 Marks Questions - Answers

A1. Different types of Android Intents with examples

1. Activity Intent (Starting new activity):

```
Intent activityIntent = new Intent(this, NewActivity.class);
activityIntent.putExtra("data", "Sample data");
startActivity(activityIntent);
```

2. Email Intent:

```
Intent emailIntent = new Intent(Intent.ACTION_SENDTO);
emailIntent.setData(Uri.parse("mailto:user@example.com"));
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Subject");
emailIntent.putExtra(Intent.EXTRA_TEXT, "Email body");
startActivity(Intent.createChooser(emailIntent, "Send Email"));
```

3. Phone Call Intent:

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:+1234567890"));
// Requires CALL_PHONE permission
startActivity(callIntent);
```

4. Web Page Intent:

```
Intent webIntent = new Intent(Intent.ACTION_VIEW);
webIntent.setData(Uri.parse("https://www.google.com"));
startActivity(webIntent);
```

A2. Intent flags and their significance

Intent flags control activity behavior and task management:

FLAG_ACTIVITY_NEW_TASK:

```
Intent intent = new Intent(this, NewActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
startActivity(intent);
```

Creates activity in a new task, useful when starting activities from non-activity contexts.

FLAG_ACTIVITY_CLEAR_TOP:

```
Intent intent = new Intent(this, MainActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
```

Clears all activities above the target activity in the task stack.

FLAG_ACTIVITY_SINGLE_TOP:

```
Intent intent = new Intent(this, CurrentActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
startActivity(intent);
```

Prevents creating new instance if activity is already at the top of the stack.

A3. Creating and handling custom actions

1. Define custom action string:

```
public static final String CUSTOM_ACTION = "com.example.app.CUSTOM_ACTION";
```

2. Register intent filter in manifest:

```
<activity android:name=".HandlerActivity">
    <intent-filter>
        <action android:name="com.example.app.CUSTOM_ACTION" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

3. Create and send custom intent:

```
Intent customIntent = new Intent(CUSTOM_ACTION);
customIntent.setType("text/plain");
customIntent.putExtra("customData", "Hello Custom Action");
startActivity(customIntent);
```

4. Handle received intent:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Intent receivedIntent = getIntent();
    if (CUSTOM_ACTION.equals(receivedIntent.getAction())) {
        String data = receivedIntent.getStringExtra("customData");
        // Process custom data
    }
}
```

A4. Security considerations with Intents

Security Risks:

- Intent spoofing: Malicious apps intercepting intents
- Data leakage: Sensitive data exposed through implicit intents
- Component hijacking: Unauthorized access to app components

Best Practices:

1. Validate incoming intents:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    Intent intent = getIntent();
    if (intent != null && isValidIntent(intent)) {
        processIntent(intent);
    }
}

private boolean isValidIntent(Intent intent) {
    return intent.getAction() != null &&
           intent.hasExtra("required_data") &&
           validateDataFormat(intent.getStringExtra("required_data"));
}
```

2. Use explicit intents for sensitive operations:

```
// Secure - explicit intent
Intent secureIntent = new Intent(this, SecureActivity.class);
startActivity(secureIntent);
```

3. Proper extra validation:

```
String userData = intent.getStringExtra("user_data");
if (userData != null && userData.length() <= MAX_LENGTH) {
    processUserData(userData);
}
```

A5. Communication using startActivityForResult()

Complete implementation for two-way communication:

Calling Activity:

```
private static final int REQUEST_CODE_USER_INPUT = 1001;

private void requestUserInput() {
    Intent intent = new Intent(this, InputActivity.class);
    intent.putExtra("prompt", "Enter your name:");
    startActivityForResult(intent, REQUEST_CODE_USER_INPUT);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == REQUEST_CODE_USER_INPUT) {
        if (resultCode == RESULT_OK && data != null) {
            String userName = data.getStringExtra("user_name");
            int userAge = data.getIntExtra("user_age", 0);

            if (isValidUserData(userName, userAge)) {
                displayUserInfo(userName, userAge);
            } else {
                showErrorMessage("Invalid user data received");
            }
        } else if (resultCode == RESULT_CANCELED) {
            showMessage("User cancelled input");
        }
    }
}

private boolean isValidUserData(String name, int age) {
    return name != null && !name.trim().isEmpty() && age > 0 && age < 150;
}
```

Input Activity:

```
private void returnResult() {
    String name = editTextName.getText().toString().trim();
    String ageStr = editTextAge.getText().toString().trim();
    if (validateInput(name, ageStr)) {
        Intent resultIntent = new Intent();
        resultIntent.putExtra("user_name", name);
        resultIntent.putExtra("user_age", Integer.parseInt(ageStr));
        setResult(RESULT_OK, resultIntent);
    } else {
        setResult(RESULT_CANCELED);
    }
    finish();
}

private boolean validateInput(String name, String ageStr) {
    return !name.isEmpty() && ageStr.matches("\\d+");
}
```

6 Marks Questions - Answers

A1. Comprehensive inter-activity communication solution

Complete Implementation Framework:

1. Custom Intent Builder Class:

```
public class IntentBuilder {
    private Intent intent;
    private Context context;

    public IntentBuilder(Context context, Class<?> targetClass) {
        this.context = context;
        this.intent = new Intent(context, targetClass);
    }

    public IntentBuilder addStringExtra(String key, String value) {
        if (value != null) intent.putExtra(key, value);
        return this;
    }

    public IntentBuilder addIntExtra(String key, int value) {
        intent.putExtra(key, value);
        return this;
    }

    public IntentBuilder addSerializableExtra(String key, Serializable object) {
        if (object != null) intent.putExtra(key, object);
        return this;
    }

    public IntentBuilder addFlags(int flags) {
        intent.addFlags(flags);
        return this;
    }

    public void start() {
        context.startActivity(intent);
    }
    public void startForResult(Activity activity, int requestCode) {
        activity.startActivityForResult(intent, requestCode);
    }
}
```

2. Data Transfer Utility:

```
public class DataTransferUtil {
    public static Bundle createBundle(Map<String, Object> data) {
        Bundle bundle = new Bundle();
        for (Map.Entry<String, Object> entry : data.entrySet()) {
            Object value = entry.getValue();
            if (value instanceof String) {
                bundle.putString(entry.getKey(), (String) value);
            } else if (value instanceof Integer) {
                bundle.putInt(entry.getKey(), (Integer) value);
            } else if (value instanceof Serializable) {
                bundle.putSerializable(entry.getKey(), (Serializable) value);
            }
        }
        return bundle;
    }

    public static <T> T extractData(Intent intent, String key, Class<T> type, T defaultValue) {
        try {
            if (type == String.class) {
                return type.cast(intent.getStringExtra(key));
            } else if (type == Integer.class) {
                return type.cast(intent.getIntExtra(key, (Integer) defaultValue));
            }
            return type.cast(intent.getSerializableExtra(key));
        } catch (Exception e) {
            return defaultValue;
        }
    }
}
```

3. Activity Result Handler:

```
public class ActivityResultHandler {
    private Map<Integer, ResultCallback> callbacks = new HashMap<>();

    public interface ResultCallback {
        void onResult(int resultCode, Intent data);
    }

    public void registerCallback(int requestCode, ResultCallback callback) {
        callbacks.put(requestCode, callback);
    }

    public void handleResult(int requestCode, int resultCode, Intent data) {
        ResultCallback callback = callbacks.get(requestCode);
        if (callback != null) {
            callback.onResult(resultCode, data);
        }
    }
}
```

4. Usage Example:

```java
// In MainActivity
private ActivityResultHandler resultHandler = new ActivityResultHandler();

private void navigateToUserProfile() {
    User user = getCurrentUser();

    new IntentBuilder(this, UserProfileActivity.class)
        .addSerializableExtra("user", user)
        .addStringExtra("mode", "edit")
        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
        .startForResult(this, REQUEST_USER_PROFILE);

    resultHandler.registerCallback(REQUEST_USER_PROFILE, (resultCode, data) -> {
        if (resultCode == RESULT_OK) {
            User updatedUser = DataTransferUtil.extractData(
                data, "updated_user", User.class, null);
            if (updatedUser != null) {
                updateUserInDatabase(updatedUser);
            }
        }
    });
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    resultHandler.handleResult(requestCode, resultCode, data);
}
```

A2. Intent filters in Android application architecture

Comprehensive Intent Filter Analysis:

1. Intent Filter Elements:

Action Element:

```xml
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.EDIT" />
    <action android:name="com.example.CUSTOM_ACTION" />
</intent-filter>
```

Category Element:

```xml
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Data Element:

```
<intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <data android:scheme="http"
      android:host="www.example.com"
      android:pathPrefix="/products"
      android:mimeType="text/html" />
</intent-filter>
```

2. Priority and Precedence:

```
<intent-filter android:priority="1000">
  <action android:name="android.intent.action.VIEW" />
  <category android:name="android.intent.category.DEFAULT" />
  <data android:scheme="myapp" />
</intent-filter>
```

3. Custom Content Provider Integration:

```
<!-- Manifest -->
<provider
  android:name=".CustomContentProvider"
  android:authorities="com.example.provider"
  android:exported="false">
  <intent-filter>
    <action android:name="android.content.action.DOCUMENTS_PROVIDER" />
  </intent-filter>
</provider>

<!-- Activity that handles provider intents -->
<activity android:name=".DocumentViewerActivity">
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="application/pdf" />
  </intent-filter>
</activity>
```

4. Best Practices:

Specific Intent Filters:

```
<!-- Good - Specific -->
<intent-filter>
  <action android:name="com.example.app.SPECIFIC_ACTION" />
  <category android:name="android.intent.category.DEFAULT" />
  <data android:mimeType="application/vnd.example.document" />
</intent-filter>

<!-- Avoid - Too broad -->
<intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <category android:name="android.intent.category.DEFAULT" />
  <data android:mimeType="*/*" />
```

```
</intent-filter>
```

5. Runtime Filter Validation:

```
public class IntentFilterValidator {
    public static boolean validateIntent(Intent intent, IntentFilter filter) {
        return filter.match(intent.getAction(),
                         intent.getType(),
                         intent.getScheme(),
                         intent.getData(),
                         intent.getCategories(),
                         "IntentFilter") >= 0;
    }
}
```

A3. Performance implications and optimization

Performance Analysis and Optimization:

1. Memory Management Issues:

```
// Problem: Large object transfer
Intent intent = new Intent(this, TargetActivity.class);
intent.putExtra("large_data", largeSerializableObject); // Memory intensive

// Solution: Use alternative approaches
public class DataCache {
    private static final Map<String, Object> cache = new HashMap<>();

    public static String storeData(Object data) {
        String key = UUID.randomUUID().toString();
        cache.put(key, data);
        return key;
    }

    public static <T> T retrieveData(String key, Class<T> type) {
        return type.cast(cache.get(key));
    }

    public static void clearData(String key) {
        cache.remove(key);
    }
}

// Usage
String dataKey = DataCache.storeData(largeObject);
intent.putExtra("data_key", dataKey);
```

2. Alternative Communication Methods:

SharedPreferences for Simple Data:

```
public class PreferencesManager {
    private static final String PREFS_NAME = "app_data";
    public static void storeString(Context context, String key, String value) {
        SharedPreferences prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
        prefs.edit().putString(key, value).apply();
    }

    public static String getString(Context context, String key, String defaultValue) {
        SharedPreferences prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
        return prefs.getString(key, defaultValue);
    }
}
```

Application Class for Global Data:

```
public class AppDataManager extends Application {
    private Map<String, Object> globalData = new HashMap<>();

    public void setGlobalData(String key, Object value) {
        globalData.put(key, value);
    }

    public <T> T getGlobalData(String key, Class<T> type) {
        return type.cast(globalData.get(key));
    }
}
```

3. Database for Complex Data:

```
public class DatabaseManager {
    private static DatabaseManager instance;
    private SQLiteDatabase database;

    public void storeComplexData(String sessionId, ComplexObject data) {
        ContentValues values = new ContentValues();
        values.put("session_id", sessionId);
        values.put("data", serializeObject(data));
        database.insert("temp_data", null, values);
    }

    public ComplexObject retrieveComplexData(String sessionId) {
        Cursor cursor = database.query("temp_data",
            new String[]{"data"},
            "session_id = ?",
            new String[]{sessionId},
            null, null, null);

        if (cursor.moveToFirst()) {
            String serializedData = cursor.getString(0);
            return deserializeObject(serializedData);
        }
        return null;
    }
}
```

4. Performance Guidelines:

- Use explicit intents for internal navigation (faster resolution)

- Avoid large Serializable objects in intents
- Implement lazy loading for complex data
- Use weak references for cached data
- Profile intent handling performance using systrace

A4. Flutter Android Integration

Complete Flutter-Android Intent Integration:

1. Platform Channel Implementation:

Android Side (MainActivity.kt):

```
class MainActivity: FlutterActivity() {
    private val INTENT_CHANNEL = "com.example.app/intents"

    override fun configureFlutterEngine(flutterEngine: FlutterEngine) {
        super.configureFlutterEngine(flutterEngine)

        MethodChannel(flutterEngine.dartExecutor.binaryMessenger, INTENT_CHANNEL)
            .setMethodCallHandler { call, result ->
                when (call.method) {
                    "sendIntent" -> {
                        val action = call.argument<String>("action")
                        val data = call.argument<Map<String, Any>>("data")
                        sendIntent(action, data, result)
                    }
                    "handleDeepLink" -> {
                        handleDeepLink(result)
                    }
                    else -> result.notImplemented()
                }
            }
    }

    private fun sendIntent(action: String?, data: Map<String, Any>?, result: MethodChannel.Result) {
        try {
            when (action) {
                "SHARE_TEXT" -> {
                    val shareText = data?.get("text") as? String
                    val shareIntent = Intent().apply {
                        this.action = Intent.ACTION_SEND
                        type = "text/plain"
                        putExtra(Intent.EXTRA_TEXT, shareText)
                    }
                    startActivity(Intent.createChooser(shareIntent, "Share via"))
                    result.success(true)
                }
                "OPEN_URL" -> {
                    val url = data?.get("url") as? String
                    val urlIntent = Intent(Intent.ACTION_VIEW, Uri.parse(url))
                    startActivity(urlIntent)
                    result.success(true)
                }
                else -> result.error("UNSUPPORTED_ACTION", "Action not supported", null)
            }
        } catch (e: Exception) {
            result.error("INTENT_ERROR", e.message, null)
        }
    }

    private fun handleDeepLink(result: MethodChannel.Result) {
        val intentData = intent.data?.toString()
        result.success(intentData)
    }
}
```

2. Flutter Side (Dart):

```
class IntentService {
  static const MethodChannel _channel = MethodChannel('com.example.app/intents');

  static Future<bool> shareText(String text) async {
    try {
      final result = await _channel.invokeMethod('sendIntent', {
        'action': 'SHARE_TEXT',
        'data': {'text': text}
      });
      return result as bool;
    } catch (e) {
      print('Error sharing text: $e');
      return false;
    }
  }

  static Future<bool> openUrl(String url) async {
    try {
      final result = await _channel.invokeMethod('sendIntent', {
        'action': 'OPEN_URL',
        'data': {'url': url}
      });
      return result as bool;
    } catch (e) {
      print('Error opening URL: $e');
      return false;
    }
  }

  static Future<String?> getDeepLinkData() async {
    try {
      final result = await _channel.invokeMethod('handleDeepLink');
      return result as String?;
    } catch (e) {
      print('Error getting deep link: $e');
      return null;
    }
  }
}
```

3. Flutter Widget Integration:

```
class IntentDemoScreen extends StatefulWidget {
  @override
  _IntentDemoScreenState createState() => _IntentDemoScreenState();
}

class _IntentDemoScreenState extends State<IntentDemoScreen> {
  String _deepLinkData = "No deep link data";

  @override
  void initState() {
    super.initState();
    _checkDeepLink();
  }

  void _checkDeepLink() async {
    final deepLinkData = await IntentService.getDeepLinkData();
    if (deepLinkData != null) {
      setState(() {
        _deepLinkData = deepLinkData;
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Intent Demo')),
      body: Column(
        children: [
          Text('Deep Link: $_deepLinkData'),
          ElevatedButton(
            onPressed: () => IntentService.shareText('Hello from Flutter!'),
            child: Text('Share Text'),
          ),
          ElevatedButton(
            onPressed: () => IntentService.openUrl('https://flutter.dev'),
            child: Text('Open URL'),
          ),
        ],
      ),
    );
  }
}
```

4. Deep Link Handling:

AndroidManifest.xml:

```
<activity
    android:name=".MainActivity"
    android:exported="true"
    android:launchMode="singleTop"
    android:theme="@style/LaunchTheme">

    <intent-filter android:autoVerify="true">
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="myapp"
              android:host="example.com" />
    </intent-filter>
</activity>
```

Error Handling and Platform Considerations:

```
class PlatformAwareIntentService {
  static Future<T?> safeInvoke<T>(String method, [dynamic arguments]) async {
    try {
      if (Platform.isAndroid) {
        return await _channel.invokeMethod<T>(method, arguments);
      } else {
        throw PlatformException(
          code: 'UNSUPPORTED_PLATFORM',
          message: 'Intent operations only supported on Android',
        );
      }
    } on PlatformException catch (e) {
      print('Platform error: ${e.message}');
      return null;
    } catch (e) {
      print('Unexpected error: $e');
      return null;
    }
  }
}
```

10 Marks Questions - Answers

A1. Comprehensive Intent Management System Design

Complete Enterprise-Level Intent Management System:

a) Architecture Design (3 marks)

Intent Router Architecture:

```
public class IntentRouter {
    private static IntentRouter instance;
    private Map<String, IntentHandler> handlers = new HashMap<>();
    private IntentValidator validator;
    private IntentLogger logger;

    public static synchronized IntentRouter getInstance() {
        if (instance == null) {
            instance = new IntentRouter();
        }
        return instance;
    }

    private IntentRouter() {
        this.validator = new IntentValidator();
        this.logger = new IntentLogger();
        initializeDefaultHandlers();
    }

    public void registerHandler(String action, IntentHandler handler) {
        handlers.put(action, handler);
        logger.logHandlerRegistration(action, handler.getClass().getSimpleName());
    }

    public boolean routeIntent(Context context, Intent intent) {
        if (!validator.validate(intent)) {
            logger.logValidationFailure(intent);
            return false;
        }

        String action = intent.getAction();
        IntentHandler handler = handlers.get(action);

        if (handler != null) {
            try {
                return handler.handle(context, intent);
            } catch (Exception e) {
                logger.logHandlingError(action, e);
                return false;
            }
        }

        return handleDefaultIntent(context, intent);
    }
}

public interface IntentHandler {
    boolean handle(Context context, Intent intent);
    String getHandlerName();
    Set<String> getSupportedActions();
}
```

Data Flow Management:

```
public class IntentDataFlowManager {
    private static final String DATA_FLOW_TAG = "IntentDataFlow";
    private DataFlowTracker tracker;

    public class DataFlowTracker {
        private Map<String, DataFlowEntry> activeFlows = new ConcurrentHashMap<>();

        public String startDataFlow(String sourceActivity, Intent intent) {
            String flowId = UUID.randomUUID().toString();
            DataFlowEntry entry = new DataFlowEntry(
                flowId, sourceActivity, intent.getAction(), System.currentTimeMillis()
            );
            activeFlows.put(flowId, entry);
            Log.d(DATA_FLOW_TAG, "Started flow: " + flowId);
            return flowId;
        }

        public void completeDataFlow(String flowId, String targetActivity, boolean success) {
            DataFlowEntry entry = activeFlows.get(flowId);
            if (entry != null) {
                entry.setTargetActivity(targetActivity);
                entry.setCompletionTime(System.currentTimeMillis());
                entry.setSuccess(success);

                // Log completion metrics
                long duration = entry.getCompletionTime() - entry.getStartTime();
                Log.d(DATA_FLOW_TAG, String.format("Flow %s completed in %dms", flowId, duration));

                // Move to historical data
                archiveFlow(entry);
                activeFlows.remove(flowId);
            }
        }
    }
}
```

Activity Lifecycle Integration:

```
public abstract class BaseIntentActivity extends AppCompatActivity {
    private IntentRouter router;
    private String currentFlowId;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        router = IntentRouter.getInstance();
        handleIncomingIntent();
    }

    private void handleIncomingIntent() {
        Intent intent = getIntent();
        if (intent != null) {
            currentFlowId = router.getDataFlowManager()
                .getTracker()
                .startDataFlow(this.getClass().getSimpleName(), intent);

            if (!router.routeIntent(this, intent)) {
                handleRoutingFailure(intent);
            }
        }
    }

    @Override
    protected void onDestroy() {
        if (currentFlowId != null) {
            router.getDataFlowManager()
                .getTracker()
                .completeDataFlow(currentFlowId, getClass().getSimpleName(), true);
        }
        super.onDestroy();
    }

    protected abstract void handleRoutingFailure(Intent intent);
}
```

b) Implementation Components (4 marks)

Custom Intent Builder with Validation:

```
public class SecureIntentBuilder {
    private Intent intent;
    private Context context;
    private Set<String> validationErrors = new HashSet<>();
    private IntentSecurityPolicy securityPolicy;

    public SecureIntentBuilder(Context context) {
        this.context = context;
        this.intent = new Intent();
        this.securityPolicy = IntentSecurityPolicy.getDefault();
    }

    public SecureIntentBuilder setAction(String action) {
        if (securityPolicy.isActionAllowed(action)) {
            intent.setAction(action);
        } else {
            validationErrors.add("Action not allowed: " + action);
        }
        return this;
    }

    public SecureIntentBuilder setComponent(ComponentName component) {
        if (securityPolicy.isComponentTrusted(component)) {
            intent.setComponent(component);
        } else {
            validationErrors.add("Untrusted component: " + component);
        }
        return this;
    }

    public SecureIntentBuilder addSecureExtra(String key, Object value) {
        if (securityPolicy.isExtraAllowed(key) &&
            securityPolicy.validateExtraValue(key, value)) {

            // Encrypt sensitive data
            if (securityPolicy.isSensitiveKey(key)) {
                value = SecurityUtils.encrypt(value.toString());
            }

            putExtraByType(key, value);
        } else {
            validationErrors.add("Extra validation failed: " + key);
        }
        return this;
    }

    public Intent buildSecure() throws IntentValidationException {
        if (!validationErrors.isEmpty()) {
            throw new IntentValidationException("Validation errors: " + validationErrors);
        }

        // Add security metadata
        intent.putExtra("_security_token", SecurityUtils.generateToken());
        intent.putExtra("_build_timestamp", System.currentTimeMillis());

        return intent;
    }

    private void putExtraByType(String key, Object value) {
        if (value instanceof String) {
            intent.putExtra(key, (String) value);
        } else if (value instanceof Integer) {
            intent.putExtra(key, (Integer) value);
        } else if (value instanceof Boolean) {
            intent.putExtra(key, (Boolean) value);
        } else if (value instanceof Serializable) {
            intent.putExtra(key, (Serializable) value);
        }
    }
}
```

Data Serialization Utility:

```
public class SecureDataSerializer {
    private static final String ENCRYPTION_KEY = "MySecretKey12345"; // In real app, use proper key
management
    private Gson gson = new Gson();

    public <T> String serializeSecurely(T object, boolean encrypt) {
        try {
            String json = gson.toJson(object);
            return encrypt ? SecurityUtils.encrypt(json) : json;
        } catch (Exception e) {
            throw new SerializationException("Failed to serialize object", e);
        }
    }

    public <T> T deserializeSecurely(String data, Class<T> clazz, boolean decrypt) {
        try {
            String json = decrypt ? SecurityUtils.decrypt(data) : data;
            return gson.fromJson(json, clazz);
        } catch (Exception e) {
            throw new SerializationException("Failed to deserialize object", e);
        }
    }

    public Bundle serializeToBundle(Object object) {
        Bundle bundle = new Bundle();

        if (object instanceof Map) {
            Map<?, ?> map = (Map<?, ?>) object;
            for (Map.Entry<?, ?> entry : map.entrySet()) {
                String key = entry.getKey().toString();
                Object value = entry.getValue();
                addToBundle(bundle, key, value);
            }
        } else {
            // Use reflection for complex objects
            Field[] fields = object.getClass().getDeclaredFields();
            for (Field field : fields) {
                field.setAccessible(true);
                try {
                    Object value = field.get(object);
                    addToBundle(bundle, field.getName(), value);
                } catch (IllegalAccessException e) {
                    Log.w("Serialization", "Could not access field: " + field.getName());
                }
            }
        }

        return bundle;
    }

    private void addToBundle(Bundle bundle, String key, Object value) {
        if (value instanceof String) {
            bundle.putString(key, (String) value);
        } else if (value instanceof Integer) {
            bundle.putInt(key, (Integer) value);
        } else if (value instanceof Boolean) {
            bundle.putBoolean(key, (Boolean) value);
        } else if (value instanceof Serializable) {
            bundle.putSerializable(key, (Serializable) value);
        }
    }
}
```

Intent Security Validator:

```
public class IntentValidator {
    private Set<String> allowedActions;
    private Set<String> trustedPackages;
    private Map<String, Pattern> dataPatterns;

    public IntentValidator() {
        initializeSecurityRules();
    }

    private void initializeSecurityRules() {
        allowedActions = new HashSet<>(Arrays.asList(
            Intent.ACTION_VIEW,
            Intent.ACTION_SEND,
            "com.example.app.CUSTOM_ACTION"
        ));

        trustedPackages = new HashSet<>(Arrays.asList(
            "com.example.app",
            "com.android.contacts"
        ));

        dataPatterns = new HashMap<>();
        dataPatterns.put("email", Pattern.compile("^[\\w.-]+@[\\w.-]+\\.[a-zA-Z]{2,}$"));
        dataPatterns.put("phone", Pattern.compile("^\\+?[1-9]\\d{1,14}$"));
        dataPatterns.put("url", Pattern.compile("^https?://[\\w.-]+\\.[a-zA-Z]{2,}.*$"));
    }
```

```
public ValidationResult validate(Intent intent) {
    ValidationResult result = new ValidationResult();

    // Validate action
    if (!isActionAllowed(intent.getAction())) {
        result.addError("Unauthorized action: " + intent.getAction());
    }

    // Validate component
    ComponentName component = intent.getComponent();
    if (component != null && !isTrustedComponent(component)) {
        result.addError("Untrusted component: " + component.getPackageName());
    }

    // Validate data
    Uri data = intent.getData();
    if (data != null && !isDataValid(data)) {
        result.addError("Invalid data URI: " + data.toString());
    }

    // Validate extras
    Bundle extras = intent.getExtras();
    if (extras != null) {
        validateExtras(extras, result);
    }

    return result;
}

private boolean isActionAllowed(String action) {
    return action != null && allowedActions.contains(action);
}

private boolean isTrustedComponent(ComponentName component) {
    return trustedPackages.contains(component.getPackageName());
}

private boolean isDataValid(Uri data) {
    String scheme = data.getScheme();
    if ("http".equals(scheme) || "https".equals(scheme)) {
        return dataPatterns.get("url").matcher(data.toString()).matches();
    }
    return true; // Allow other schemes for now
}

private void validateExtras(Bundle extras, ValidationResult result) {
    for (String key : extras.keySet()) {
        Object value = extras.get(key);

        // Check for sensitive keys
        if (isSensitiveKey(key) && !isValueSecure(value)) {
            result.addWarning("Potentially sensitive data in extra: " + key);
        }

        // Validate specific data types
        if ("email".equals(key) && value instanceof String) {
                if (!dataPatterns.get("email").matcher((String) value).matches()) {
                    result.addError("Invalid email format: " + value);
                }
            }
        }
    }

    private boolean isSensitiveKey(String key) {
        return key.toLowerCase().contains("password") ||
               key.toLowerCase().contains("token") ||
               key.toLowerCase().contains("secret");
    }

    private boolean isValueSecure(Object value) {
        // Check if value appears to be encrypted (simplified check)
        return value instanceof String &&
               ((String) value).length() > 50 &&
               !((String) value).contains(" ");
    }
}
```

Error Handling Framework:

```
public class IntentErrorHandler {
    private static final String TAG = "IntentError";
    private List<ErrorListener> errorListeners = new ArrayList<>();

    public interface ErrorListener {
        void onIntentError(IntentError error);
    }

    public void handleError(IntentError error) {
        // Log error
        Log.e(TAG, "Intent error: " + error.getMessage(), error.getCause());

        // Report to analytics
        reportToAnalytics(error);

        // Notify listeners
        for (ErrorListener listener : errorListeners) {
            try {
                listener.onIntentError(error);
            } catch (Exception e) {
                Log.e(TAG, "Error in error listener", e);
            }
        }

        // Handle specific error types
        switch (error.getType()) {
            case VALIDATION_ERROR:
                handleValidationError(error);
                break;
            case SECURITY_ERROR:
                handleSecurityError(error);
                break;
            case ROUTING_ERROR:
                handleRoutingError(error);
                break;
            default:
                handleGenericError(error);
        }
    }

    private void handleValidationError(IntentError error) {
        // Show user-friendly message
        showErrorDialog("Invalid request", "The requested action could not be completed due to invalid data.");
    }

    private void handleSecurityError(IntentError error) {
        // Log security incident
        SecurityLogger.logSecurityIncident(error);
        showErrorDialog("Security Error", "The request was blocked for security reasons.");
    }

    private void handleRoutingError(IntentError error) {
        // Attempt fallback routing
        if (!attemptFallbackRouting(error.getIntent())) {
            showErrorDialog("Service Unavailable", "The requested service is not available.");
        }
    }
}
```

c) Advanced Features (3 marks)

Deep Linking Implementation:

```
public class DeepLinkManager {
    private static final String DEEP_LINK_SCHEME = "myapp";
    private Map<String, DeepLinkHandler> linkHandlers = new HashMap<>();
    private DeepLinkAnalytics analytics;

    public void registerDeepLinkHandler(String path, DeepLinkHandler handler) {
        linkHandlers.put(path, handler);
    }

    public boolean handleDeepLink(Context context, Uri uri) {
        if (!DEEP_LINK_SCHEME.equals(uri.getScheme())) {
            return false;
        }

        String path = uri.getPath();
        DeepLinkHandler handler = findHandler(path);

        if (handler != null) {
            try {
                Map<String, String> params = extractParameters(uri);
                boolean success = handler.handle(context, path, params);

                analytics.trackDeepLink(uri.toString(), success);
                return success;
            } catch (Exception e) {
                Log.e("DeepLink", "Error handling deep link", e);
                analytics.trackDeepLinkError(uri.toString(), e.getMessage());
                return false;
            }
        }

        return handleDefaultDeepLink(context, uri);
    }

    private DeepLinkHandler findHandler(String path) {
        // Exact match first
        if (linkHandlers.containsKey(path)) {
            return linkHandlers.get(path);
        }

        // Pattern matching for parameterized paths
        for (Map.Entry<String, DeepLinkHandler> entry : linkHandlers.entrySet()) {
            if (pathMatches(entry.getKey(), path)) {
                return entry.getValue();
            }
        }

        return null;
    }

    private boolean pathMatches(String pattern, String path) {
        // Simple pattern matching (can be enhanced with regex)
        String[] patternParts = pattern.split("/");
        String[] pathParts = path.split("/");

        if (patternParts.length != pathParts.length) {
            return false;
        }

        for (int i = 0; i < patternParts.length; i++) {
            if (!patternParts[i].startsWith("{") && !patternParts[i].equals(pathParts[i])) {
                return false;
            }
        }

        return true;
    }

    public interface DeepLinkHandler {
        boolean handle(Context context, String path, Map<String, String> params);
    }
}
```

Intent-Based Navigation Patterns:

```
public class NavigationManager {
    private static NavigationManager instance;
    private Stack<NavigationEntry> navigationStack = new Stack<>();
    private Map<String, Intent> savedIntents = new HashMap<>();

    public static NavigationManager getInstance() {
        if (instance == null) {
            instance = new NavigationManager();
        }
        return instance;
    }

    public void navigateWithHistory(Context context, Intent intent) {
        // Save current state
        if (context instanceof Activity) {
            NavigationEntry entry = new NavigationEntry(
                context.getClass().getSimpleName(),
                System.currentTimeMillis(),
                captureActivityState((Activity) context)
            );
            navigationStack.push(entry);
        }

        // Navigate to new activity
        context.startActivity(intent);
    }

    public boolean navigateBack(Activity currentActivity) {
        if (navigationStack.isEmpty()) {
            return false;
        }

        NavigationEntry previousEntry = navigationStack.pop();

        // Create intent to return to previous activity
        try {
            Class<?> activityClass = Class.forName(previousEntry.getActivityClassName());
            Intent backIntent = new Intent(currentActivity, activityClass);

            // Restore previous state if available
            if (previousEntry.getState() != null) {
                backIntent.putExtras(previousEntry.getState());
            }

            backIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_SINGLE_TOP);
            currentActivity.startActivity(backIntent);
            currentActivity.finish();

            return true;
        } catch (ClassNotFoundException e) {
            Log.e("Navigation", "Could not find previous activity class", e);
            return false;
        }
    }

    public void saveIntentForLater(String key, Intent intent) {
        savedIntents.put(key, intent);
    }

    public Intent getSavedIntent(String key) {
        return savedIntents.remove(key);
    }

    private Bundle captureActivityState(Activity activity) {
        Bundle state = new Bundle();

        // Capture basic activity information
        state.putString("activity_title",
            activity.getTitle() != null ? activity.getTitle().toString() : "");

        // Let activity save its own state
        if (activity instanceof StatefulActivity) {
            Bundle activityState = ((StatefulActivity) activity).saveState();
            state.putBundle("activity_state", activityState);
        }

        return state;
    }

    public interface StatefulActivity {
        Bundle saveState();
        void restoreState(Bundle state);
    }
}
```

Performance Optimization Framework:

```
public class IntentPerformanceOptimizer {
    private static final int MAX_CACHE_SIZE = 100;
    private LRUCache<String, Intent> intentCache;
    private IntentMetrics metrics;

    public IntentPerformanceOptimizer() {
        intentCache = new LRUCache<String, Intent>(MAX_CACHE_SIZE) {
            @Override
            protected void entryRemoved(boolean evicted, String key, Intent oldValue, Intent newValue) {
                if (evicted) {
                    metrics.recordCacheEviction(key);
                }
            }
        };
        metrics = new IntentMetrics();
    }

    public Intent getCachedIntent(String key) {
        long startTime = System.nanoTime();
        Intent cached = intentCache.get(key);
        long endTime = System.nanoTime();

        metrics.recordCacheAccess(key, cached != null, endTime - startTime);
        return cached;
    }

    public void cacheIntent(String key, Intent intent) {
        // Only cache if intent is reusable and not too large
        if (isIntentCacheable(intent)) {
            intentCache.put(key, cloneIntent(intent));
            metrics.recordCacheStore(key);
        }
    }
    private boolean isIntentCacheable(Intent intent) {
        // Don't cache intents with large extras
        Bundle extras = intent.getExtras();
        if (extras != null) {
            Parcel parcel = Parcel.obtain();
            try {
                extras.writeToParcel(parcel, 0);
                int size = parcel.dataSize();
                return size < 10 * 1024; // 10KB limit
            } finally {
                parcel.recycle();
            }
        }
        return true;
    }

    private Intent cloneIntent(Intent original) {
        Intent clone = new Intent(original);
        // Deep copy extras if needed
        Bundle originalExtras = original.getExtras();
        if (originalExtras != null) {
            clone.putExtras(new Bundle(originalExtras));
        }
        return clone;
    }

    public void optimizeIntentForPerformance(Intent intent) {
        // Remove unnecessary flags
        intent.removeFlags(Intent.FLAG_ACTIVITY_NO_ANIMATION);

        // Compress large string extras
        Bundle extras = intent.getExtras();
        if (extras != null) {
            compressLargeExtras(extras);
        }
    }

    private void compressLargeExtras(Bundle extras) {
        for (String key : extras.keySet()) {
            Object value = extras.get(key);
            if (value instanceof String) {
                String stringValue = (String) value;
                if (stringValue.length() > 1000) {
                    // Compress large strings
                    String compressed = CompressionUtils.compress(stringValue);
                    extras.putString(key, compressed);
                    extras.putBoolean(key + "_compressed", true);
                }
            }
        }
    }
}
```

This comprehensive intent management system provides enterprise-level functionality with security, performance optimization, error handling, and advanced navigation patterns. The system is designed to be scalable, maintainable, and secure for production applications.

A2. Cross-Platform Intent Integration: Android and Flutter

Complete Flutter-Android Intent Integration Solution:

a) Platform Channel Architecture (3 marks)

Method Channel Implementation:

Android Side - IntentMethodChannel.kt:

```
class IntentMethodChannel(private val activity: FlutterActivity) {
    companion object {
        private const val CHANNEL_NAME = "com.example.app/intent_manager"
        private const val EVENT_CHANNEL_NAME = "com.example.app/intent_events"
    }

    private lateinit var methodChannel: MethodChannel
    private lateinit var eventChannel: EventChannel
    private var eventSink: EventChannel.EventSink? = null
    private val intentQueue = ArrayDeque<Map<String, Any>>()

    fun setupChannels(flutterEngine: FlutterEngine) {
        methodChannel = MethodChannel(
            flutterEngine.dartExecutor.binaryMessenger,
            CHANNEL_NAME
        )
        methodChannel.setMethodCallHandler(::handleMethodCall)

        eventChannel = EventChannel(
            flutterEngine.dartExecutor.binaryMessenger,
            EVENT_CHANNEL_NAME
        )
        eventChannel.setStreamHandler(object : EventChannel.StreamHandler {
            override fun onListen(arguments: Any?, events: EventChannel.EventSink?) {
                eventSink = events
                // Send queued intents
                while (intentQueue.isNotEmpty()) {
                    eventSink?.success(intentQueue.removeFirst())
                }
            }

            override fun onCancel(arguments: Any?) {
                eventSink = null
            }
        })
    }

    private fun handleMethodCall(call: MethodCall, result: MethodChannel.Result) {
        when (call.method) {
            "sendIntent" -> handleSendIntent(call, result)
            "startActivityForResult" -> handleStartActivityForResult(call, result)
            "shareContent" -> handleShareContent(call, result)
            "openUrl" -> handleOpenUrl(call, result)
            "pickFile" -> handlePickFile(call, result)
            "takePhoto" -> handleTakePhoto(call, result)
            "getInstalledApps" -> handleGetInstalledApps(result)
```

```kotlin
            else -> result.notImplemented()
        }
    }

    private fun handleSendIntent(call: MethodCall, result: MethodChannel.Result) {
        try {
            val action = call.argument<String>("action") ?: return result.error(
                "INVALID_ARGUMENT", "Action is required", null
            )
            val extras = call.argument<Map<String, Any>>("extras") ?: emptyMap()
            val flags = call.argument<List<String>>("flags") ?: emptyList()

            val intent = Intent(action).apply {
                // Add extras with proper type conversion
                extras.forEach { (key, value) ->
                    when (value) {
                        is String -> putExtra(key, value)
                        is Int -> putExtra(key, value)
                        is Double -> putExtra(key, value.toFloat())
                        is Boolean -> putExtra(key, value)
                        is List<*> -> {
                            if (value.all { it is String }) {
                                putStringArrayListExtra(key, ArrayList(value as List<String>))
                            }
                        }
                    }
                }

                // Add flags
                flags.forEach { flag ->
                    when (flag) {
                        "FLAG_ACTIVITY_NEW_TASK" -> addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
                        "FLAG_ACTIVITY_CLEAR_TOP" -> addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
                        "FLAG_ACTIVITY_SINGLE_TOP" -> addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP)
                    }
                }
            }

            activity.startActivity(intent)
            result.success(true)

        } catch (e: Exception) {
            result.error("INTENT_ERROR", e.message, null)
        }
    }

    private fun handleShareContent(call: MethodCall, result: MethodChannel.Result) {
        try {
            val text = call.argument<String>("text")
            val subject = call.argument<String>("subject")
            val mimeType = call.argument<String>("mimeType") ?: "text/plain"

            val shareIntent = Intent().apply {
                action = Intent.ACTION_SEND
                type = mimeType
                text?.let { putExtra(Intent.EXTRA_TEXT, it) }
                subject?.let { putExtra(Intent.EXTRA_SUBJECT, it) }
            }

            activity.startActivity(Intent.createChooser(shareIntent, "Share via"))
            result.success(true)

        } catch (e: Exception) {
            result.error("SHARE_ERROR", e.message, null)
        }
    }

    // Handle incoming intents and broadcast to Flutter
    fun handleIncomingIntent(intent: Intent) {
        val intentData = mapOf(
            "action" to intent.action,
            "data" to intent.dataString,
            "extras" to extractExtras(intent),
            "timestamp" to System.currentTimeMillis()
        )

        if (eventSink != null) {
            eventSink?.success(intentData)
        } else {
            // Queue intent if Flutter isn't ready
            intentQueue.offer(intentData)
        }
    }

    private fun extractExtras(intent: Intent): Map<String, Any> {
        val extras = mutableMapOf<String, Any>()
        intent.extras?.let { bundle ->
            bundle.keySet().forEach { key ->
                bundle.get(key)?.let { value ->
                    when (value) {
                        is String, is Int, is Double, is Boolean -> extras[key] = value
                        is ArrayList<*> -> {
                            if (value.all { it is String }) {
                                extras[key] = value
                            }
                        }
                    }
                }
            }
        }
        return extras
    }
}
```

Flutter Side - IntentManager.dart:

```
class IntentManager {
  static const MethodChannel _methodChannel =
      MethodChannel('com.example.app/intent_manager');
  static const EventChannel _eventChannel =
      EventChannel('com.example.app/intent_events');

  static Stream<Map<String, dynamic>>? _intentStream;
static final List<IntentListener> _listeners = [];

// Method channel operations
static Future<bool> sendIntent({
  required String action,
  Map<String, dynamic>? extras,
  List<String>? flags,
}) async {
  try {
    final result = await _methodChannel.invokeMethod('sendIntent', {
      'action': action,
      'extras': extras ?? {},
      'flags': flags ?? [],
    });
    return result as bool;
  } catch (e) {
    print('Error sending intent: $e');
    return false;
  }
}

static Future<bool> shareContent({
  String? text,
  String? subject,
  String mimeType = 'text/plain',
}) async {
  try {
    final result = await _methodChannel.invokeMethod('shareContent', {
      'text': text,
      'subject': subject,
      'mimeType': mimeType,
    });
    return result as bool;
  } catch (e) {
    print('Error sharing content: $e');
    return false;
  }
}

static Future<bool> openUrl(String url) async {
  return await sendIntent(
    action: 'android.intent.action.VIEW',
    extras: {'android.intent.extra.TEXT': url},
  );
}

static Future<List<AppInfo>> getInstalledApps() async {
  try {
    final result = await _methodChannel.invokeMethod('getInstalledApps');
    final List<dynamic> apps = result as List<dynamic>;
    return apps.map((app) => AppInfo.fromMap(app)).toList();
  } catch (e) {
    print('Error getting installed apps: $e');
    return [];
  }
}
```

```
  // Event channel for incoming intents
  static Stream<Map<String, dynamic>> get intentStream {
    _intentStream ??= _eventChannel
        .receiveBroadcastStream()
        .map((event) => Map<String, dynamic>.from(event));
    return _intentStream!;
  }

  static void addIntentListener(IntentListener listener) {
    if (!_listeners.contains(listener)) {
      _listeners.add(listener);

      // Start listening if this is the first listener
      if (_listeners.length == 1) {
        _startListening();
      }
    }
  }

  static void removeIntentListener(IntentListener listener) {
    _listeners.remove(listener);

    // Stop listening if no more listeners
    if (_listeners.isEmpty()) {
      _stopListening();
    }
  }

  static void _startListening() {
    intentStream.listen((intentData) {
      final intent = IncomingIntent.fromMap(intentData);
      for (final listener in _listeners) {
        try {
          listener.onIntentReceived(intent);
        } catch (e) {
          print('Error in intent listener: $e');
        }
      }
    });
  }

  static void _stopListening() {
    // Event channel streams are automatically managed
    // We just clear our listener references
  }
}

// Data models
class AppInfo {
  final String packageName;
  final String appName;
```

Note:

Some questions are not a part of CIE 1 (like intent with flutter, flags related), Take those which are related to CIE 1 , others questions for knowledge.

Thank you