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 - 1	**Topic:The UI Thread, Threads and Runnables**
Editor: 1RV24MC108 – Swastika Devadiga	

List of Questions

1. **What is the purpose of the run() method in a thread?**
2. **Mention two ways to create thread in java.**
3. **What is thread in android?**
4. **What is Runnable interface?**
5. **Explain how to create thread using the Thread class with example**
6. **How does the Runnable interface differ from extending Thread?**
7. **Differentiate between Extending Thread and Implementing Thread.**
8. **Write a short note on View binding and its advantages**
9. **Explain the steps to perform a network operation using OkHttpClient in a background Thread.**
10. **Implement a GitHub User Info Fetcher Using Threads in Android**
11. **Discuss the Importance of multithreading in Android Development with Examples**

02 Marks Questions:

Q: **What is the purpose of the run() method in a thread?**
Answer:
The run() method contains the code that should execute in a separate thread. When a thread is started using start(), it internally calls the run() method.

Q:**Mention two ways to create thread in java.**
Answer:

1.By extending the Thread class.

2.By implementing the Runnable interface.

Q:**What is thread in android?**
Answer:
A thread in Android is a unit of execution within a process. It allows for concurrent

operations,enabling tasks like network requests or file I/O to run in the background without blocking the main UI thread.

Q:**What is Runnable interface?**
Answer:
The Runnable interface in Java represents a task that can be executed by a thread. It contains a single method, run(), which defines the code to be executed. Implementing Runnable allows for defining tasks to be run on separate threads.

04 Marks Questions:

Q: **Explain how to create thread using the Thread class with example**
Answer:
It can be achieved by the Extending the Thread class.

Extend the Thread class.
Override the run() method.
Create an object and call start().

Example:

```
class Worker extends Thread {

        public void run() {

        // background task

        }

    }

    Worker worker = new Worker();
     worker.start();
```

Q:**How does the Runnable interface differ from extending Thread?**
Answer:
With Runnable, you implement the run() method and pass the instance to a Thread object.

This is more flexible than extending Thread as it allows multiple inheritance and better object sharing.

```
class Worker implements Runnable {
public void run() {
        // background task
    }
}
```

Worker worker = new Worker();

Thread thread = new Thread(worker);

thread.start();

06 Marks Questions:

Q:**Differentiate between Extending Thread and Implementing Thread.**

Answer:

Aspect	**Extending Thread**	**Implementing Runnable**
Inheritance	Subclass of `Thread`	Implements `Runnable` interface
Flexibility	Cannot extend other classes	Can extend other classes
Code Reusability	Less reusable	More reusable and decoupled
Instantiation	`Worker worker = new Worker();worker.start();`	`Worker worker = new Worker();Thread thread = new Thread(worker);thread.start();`

Q:**Write a short note on View binding and its advantages**
Answer:

- View Binding is a feature that allows safer and easier access to UI components in XML.
- When enabled, it generates a binding class for each layout file.
- The generated class directly references all views with an ID, avoiding `findViewById`.
-

Example:
If XML is `activity_main.xml`, the generated class is `ActivityMainBinding`, and we can access views like:

```
binding.textView.setText("Hello");
```

Advantages:

1. Type-safe.
2. No need for casting.
3. Reduces boilerplate code.
4. Improves compile-time safety.

08 Marks Questions

Q: **Explain the steps to perform a network operation using OkHttpClient in a background Thread.**
Answer:

1. Accept GitHub ID as parameter.
2. Construct URL: `https://api.github.com/users/{gitHubId}`
3. Initialize OkHttpClient.
4. Make GET request and receive response.
5. Return the result.

Code Example:

```
private String fetchUserInfo(String gitHubId) {
  String url = String.format("https://api.github.com/users/%s", gitHubId);
  OkHttpClient client = new OkHttpClient();
  Request request = new Request.Builder().url(url).build();

  try (Response response = client.newCall(request).execute()) {
        return response.body().string();
  } catch (IOException e) {
        Log.e("TAG", e.getMessage());
        return "";
  }
}
```

This must be called in a background thread to avoid `NetworkOnMainThreadException`

10 Marks Questions

Q: **Implement a GitHub User Info Fetcher Using Threads in Android**
Answer:
Concepts Involved:

UI Thread (Main Thread):

Handles all user interface and interaction tasks. Blocking this thread (e.g., by performing network operations) can freeze the app or cause ANR (Application Not Responding) errors.

Background Thread:

Used to perform long-running operations (like API calls or database access) without blocking the UI thread.

Thread Class:

Java's built-in Thread class can be used to offload tasks from the UI thread.
Handler or runOnUiThread():
Used to update UI components from a background thread.

How it Works:

1.User enters a GitHub username.
2.A background thread makes a network call to the GitHub API.
3.UI is updated with the fetched data on the main thread.

Permissions (AndroidManifest.xml):

Add internet permission:

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

Dependencies (build.gradle):

Include OkHttp for network operations:
implementation 'com.squareup.okhttp3:okhttp:4.7.2'

GitHub API Used:

https://api.github.com/users/{username}

Example:

https://api.github.com/users/octocat

Key Components Used:

EditText – for user input

Button – to trigger fetch operation

TextView – to display results

Thread – to perform background fetch

Handler or runOnUiThread() – to safely update UI

Sample Code Snippet:

```
public class MainActivity extends AppCompatActivity {
        private TextView resultView;
        private EditText usernameInput;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        resultView = findViewById(R.id.resultView);
        usernameInput = findViewById(R.id.usernameInput);
        Button fetchButton = findViewById(R.id.fetchButton);

        fetchButton.setOnClickListener(v -> {
        String username = usernameInput.getText().toString();
        new Thread(() -> {
                String userInfo = fetchUserInfo(username);
                runOnUiThread(() -> resultView.setText(userInfo));
        }).start();
        });
}


        private String fetchUserInfo(String username) {
        OkHttpClient client = new OkHttpClient();
        Request request = new Request.Builder()
        .url("https://api.github.com/users/" + username)
        .build();
        try (Response response = client.newCall(request).execute()) {
        return response.body().string();
        } catch (IOException e) {
        return "Error: " + e.getMessage();
        }
        }
}
```

Q:**Discuss the Importance of multithreading in Android Development with Examples.**
Answer:
Importance of Multithreading in Android:
Multithreading is a crucial concept in Android development. It allows an application to perform multiple operations simultaneously, improving performance and responsiveness.
In Android, the main thread (UI thread) is responsible for handling user interactions and rendering UI elements.
If any time-consuming operation is performed on this thread, the app may freeze or become unresponsive.

Key Reasons Why Multithreading is Important:

1.Responsive UI:

Time-consuming tasks like file reading, database queries, or API calls can freeze the app if done on the main thread.

Multithreading keeps the UI thread free to handle user inputs and redraw the screen smoothly.

2.Efficient CPU Utilization:

Threads allow parallel execution of tasks, leading to better use of multi-core processors.

3.Faster Performance:

Background tasks like image processing, downloading data, and complex calculations can run without interrupting user interaction.

4.Avoiding ANR (Application Not Responding) Errors:

Android will show an ANR dialog if the main thread is blocked for more than 5 seconds. Using threads prevents such errors.

Examples:

Image Loading: Loading images in a background thread to prevent UI lag.

Database Access: Performing database queries in a separate thread to avoid blocking the main thread.

Network Requests: Fetching data from APIs using background threads and updating the UI upon completion.

Small Real-Life Example:

Let’s say a user enters a GitHub username and taps a button.A background thread fetches user info from the GitHub API.After receiving the response, the UI thread updates a TextView to show followers, following, and profile details.This use of multithreading makes the app responsive and smooth, even during slow network conditions.