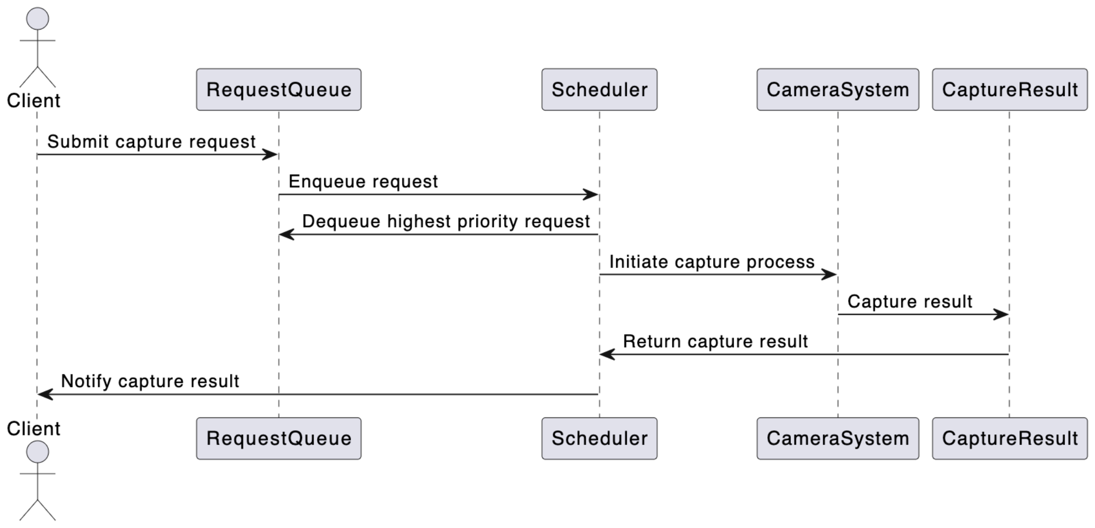
Concurrent Capture System - LLD

Low level design:

Classes:

1. Camera
2. Client request
3. Request queue
4. Request processor
5. Scheduler

Use case diagram:



Flow :

1. Two requests are submitted —> Request A (Urgency Level 5) and Request B (Urgency Level 9).
2. Both requests are added to the RequestQueue. The queue prioritizes Request B due to its higher urgency level.
3. The Scheduler periodically checks the queue and retrieves Request B first for processing.
4. The RequestProcessor handles Request B using the Camera and processes its action. After Request B is completed, Request A is processed similarly.

5. Requests are processed based on priority, with high-urgency requests handled before lower-urgency ones.

LLD logic:

```
@Getter
@Setter
public class Camera {

    private final String id;
    private final String name;
    private final String model;
    private boolean isRecording = false;

    public Camera(String id, String name, String model) {
        this.id = id;
        this.name = name;
        this.model = model;
    }

    public synchronized void startRecording() {
        // Start recording logic
    }

    public synchronized void stopRecording() {
        // Stop recording logic
    }

    public void captureImage() {
        // Capture image logic
    }
}

public class RequestProcessor implements IRequestProcessor {

    private final Camera camera;
    private final RequestQueue requestQueue;
    private final ExecutorService executorService;

    public RequestProcessor(Camera camera, RequestQueue requestQueue, int
poolSize) {
        this.camera = camera;
```

```java
        this.requestQueue = requestQueue;
        this.executorService = Executors.newFixedThreadPool(poolSize);
        startProcessing();
    }

    private void startProcessing() {
        // Start processing requests
    }

    public void handleRequest(Request request) {
        // Add request to queue
    }

    public void shutdown() {
        // Shutdown the executor service
    }
}
```

```java
public class RequestQueue implements IRequestQueue {

    private final PriorityBlockingQueue<Request> queue;

    public RequestQueue() {
        queue = new PriorityBlockingQueue<>(
            11, (r1, r2) -> Integer.compare(r2.getUrgencyLevel(),
r1.getUrgencyLevel()) // Priority based on urgency
        );
    }

    public void offer(Request request) {
        // Add request to queue
    }

    public Request take() throws InterruptedException {
        return null;
    }
}
```

```
public class Scheduler implements IScheduler {
    private final RequestQueue requestQueue;
    private final ScheduledExecutorService scheduler;

    public Scheduler(RequestQueue requestQueue) {
        this.requestQueue = requestQueue;
        this.scheduler = Executors.newScheduledThreadPool(1);
    }

    public void startScheduling() {
        // Schedule task to periodically process requests
        scheduler.scheduleAtFixedRate(() -> {
            try {
                Request request = requestQueue.take();
                // Handle the request using a separate processing mechanism
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }, 0, 1, TimeUnit.SECONDS); // Adjust interval as needed
    }


    public void shutdown() {
        scheduler.shutdown();
    }
}
```

```
public interface IScheduler {
    void startScheduling();
    void shutdown();
}

public interface IRequestProcessor {
    void handleRequest(IRequest request);
    void shutdown();
}

public interface IRequestQueue {
    void offer(IRequest request);
    IRequest take() throws InterruptedException;
}
```