Great! Let’s go through both:

**✅ 1) Semaphore in Java**

**🔹 What is a Semaphore?**

A Semaphore is a **concurrency utility** that controls access to a **limited number of permits** (like tickets).

Think of it as a **door with limited entries** – only a fixed number of threads can enter at a time.

**🔹 Use case:**

* Limit concurrent access to resources (e.g., max 3 threads can write to a DB).
* Managing thread pools manually.
* Rate limiting.

**🔹 Example:**

import java.util.concurrent.Semaphore;

public class PrintJobManager {

private static final Semaphore semaphore = new Semaphore(3); // 3 permits

public void printJob(String jobName) {

try {

semaphore.acquire(); // get a permit (wait if none)

System.out.println(jobName + " is printing...");

Thread.sleep(2000);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

} finally {

semaphore.release(); // give back the permit

System.out.println(jobName + " is done.");

}

}

public static void main(String[] args) {

PrintJobManager manager = new PrintJobManager();

for (int i = 1; i <= 6; i++) {

String job = "Job-" + i;

new Thread(() -> manager.printJob(job)).start();

}

}

}

**🔹 Output Behavior:**

Only 3 threads (jobs) run in parallel. Others wait.

**✅ 2) ArrayBlockingQueue**

**🔹 What is it?**

ArrayBlockingQueue is a **bounded**, **thread-safe queue** backed by an array. It blocks threads trying to **add to a full queue** or **remove from an empty queue**.

**🔹 Use case:**

* **Producer-Consumer** scenarios
* Thread communication between services
* Rate limiting / task queues

**🔹 Example:**

import java.util.concurrent.\*;

public class ProducerConsumer {

private static BlockingQueue<Integer> queue = new ArrayBlockingQueue<>(3); // capacity 3

public static void main(String[] args) {

Runnable producer = () -> {

int value = 1;

try {

while (true) {

queue.put(value); // blocks if full

System.out.println("Produced: " + value);

value++;

Thread.sleep(500);

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

};

Runnable consumer = () -> {

try {

while (true) {

Integer val = queue.take(); // blocks if empty

System.out.println("Consumed: " + val);

Thread.sleep(1000);

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

};

new Thread(producer).start();

new Thread(consumer).start();

}

}

**🔹 Blocking methods:**

* put(E) → blocks if full
* take() → blocks if empty

**🔄 Comparison Quick View:**

| **Feature** | **Semaphore** | **ArrayBlockingQueue** |
| --- | --- | --- |
| Controls | Access to resource (permits) | Queue size (items) |
| Blocking? | Yes – on acquire() | Yes – on put()/take() |
| Use case | Limit concurrent access | Producer-Consumer queue |
| Backed by | Internal counter | Array (fixed-size) |
| Thread-safe | ✅ | ✅ |

Do you want to see a Semaphore example with real-time DB connection pool simulation or deeper use of BlockingQueue in multithreaded architecture like logging service?