Hard: Handling Partial Failures with `Promise.allSettled`

Objective: Understand how to manage multiple promises when some may fail, using `Promise.allSettled`.

Task: Write a function named `fetchMultipleResources` that accepts an array of URLs (simulated as asynchronous functions that return Promises). This function should:

1. Attempt to fetch data from all given URLs concurrently, acknowledging that some may fail.
2. Use `Promise.allSettled` to wait for all operations to complete, regardless of whether they succeed or fail.
3. Filter the results to separate the successfully fetched data from the errors.
4. Return an object with two properties: `successes` containing all successful fetch results, and `errors` listing the reasons for any failures.

Criteria for Success:

- Implement the use of `Promise.allSettled` correctly to handle all given promises.
- Correctly process and separate successful results from failures.
- Demonstrate the functionality with a mix of promises that resolve and reject.

Expert: Dynamic Promise Retry Mechanism

Objective: Implement a retry mechanism for Promises that handles failure by retrying a specified number of times before giving up.

Task: Create a function named `retryPromise` that:

1. Accepts a function that returns a Promise, a maximum number of retries, and a delay between retries.
2. Attempts to execute the given Promise. If the Promise fails (rejects), it retries the operation after the specified delay, up to the maximum number of retries.
3. If all attempts fail, the function should finally reject with the last error encountered.
4. If any attempt succeeds, resolve immediately with the successful result.

Criteria for Success:

- Correct implementation of retry logic with delay handling between attempts.
- Proper error handling, ensuring the last error is propagated if all retries fail.
- Validation through testing with a promise that randomly fails or succeeds, demonstrating the retry mechanism.

Advanced: Coordinated Parallel Tasks with Dependency Resolution

Objective: Execute multiple asynchronous tasks in parallel, some of which depend on the completion of others, using `Promise.all` and careful orchestration.

Task: Imagine a scenario where you have four tasks (A, B, C, D), with B depending on A, and D depending on both B and C. Write a function `executeTasks` that:

1. Executes tasks A, B, C, and D, respecting their dependencies. Tasks without dependencies (A, C) should start immediately, B should wait for A, and D should wait for both B and C.
2. Each task is simulated by a function that returns a Promise, resolving after a random delay.
3. Utilize `Promise.all` or other appropriate promise methods to manage the dependencies and execution order efficiently.
4. Log the result of each task upon completion, ensuring the correct execution order is maintained.

Criteria for Success:

- Efficient use of Promises to manage and synchronize tasks with dependencies.
- Correct execution order that respects the given dependencies.
- Demonstrated handling of asynchronous task completion with appropriate logging.