Apache Spark Java Optimization Cheat Sheet

Use mapPartitions for Efficiency

```
// Avoid map() overhead on large datasets
JavaRDD<String> optimized = rdd.mapPartitions(iterator -> {
    List<String> results = new ArrayList<>();
    while (iterator.hasNext()) results.add(process(iterator.next()));
    return results.iterator();
});
```

Filter Before Exploding Arrays

```
// Prevent explosion of irrelevant nested data
df.filter("array_contains(tags, 'important')")
  .selectExpr("explode(tags) as tag");
```

Repartition to Control Output Files

```
// Prevent too many small/large output files
df.repartition(8).write().parquet("output/optimized");
```

Select Specific Columns Instead of *

```
// Avoid reading unneeded columns
df.select("id", "name", "created_date");
```

Combine withColumn Logic in One Select

```
// Avoid chaining multiple withColumn calls
df.select(
    col("name"),
    functions.upper(col("name")).alias("upper_name"),
    functions.length(col("name")).alias("name_length")
);
```

Broadcast Small Lookup Tables

```
Broadcast<Map<String, String>> bMap = sc.broadcast(lookupData);
rdd.map(row -> bMap.value().get(row.split(",")[0]));
```

Avoid groupByKey in Aggregations

```
// Use reduceByKey to prevent memory issues
rdd.reduceByKey((a, b) -> a + b);
```

Use Window Functions Over Joins

```
WindowSpec w = Window.partitionBy("user_id").orderBy(col("timestamp").desc());
df.withColumn("rank", row_number().over(w)).filter("rank = 1");
```

Cache Reused DataFrames

```
Dataset<Row> enriched = df.join(dim, "id").cache();
enriched.count(); // triggers cache
```

Filter Early in the Pipeline

Apache Spark Java Optimization Cheat Sheet

```
// Filter before joining or grouping
df.filter("age > 30").join(otherDf, "id");
```