Hedging strategy **CHEAT SHEET**

Basics

This reactive resilience strategy **allows you to execute more than one invocation** if the original has failed or considered too slow.

You can configure the behaviour of the strategy via the **HedgingStrategyOptions<T>** object.

The execution can be configured to run the invocations *sequentially* or *concurrently*.

Set the **Delay greater than 0 seconds** to run the invocations *sequentially* and issue a new one only if the previous one is taking too long.

Set the **Delay less than 0 second** to run the invocations *sequentially* and issue a new one only if the previous one failed.

Set the **Delay to 0 seconds** to run the invocations *concurrently* and wait only for the fastest one to complete.

Specify sequential retries for slow execution

```
new ResiliencePipelineBuilder<int>()
  .AddHedging(new HedgingStrategyOptions<int>()
  {
    Delay = TimeSpan.FromSeconds(1),
    MaxHedgedAttempts = 2
  })
```

Specify sequential retries for failed execution + notification

```
new ResiliencePipelineBuilder<int>()
  .AddHedging(new HedgingStrategyOptions<int>()
  {
    Delay = TimeSpan.FromSeconds(-1),
    ShouldHandle = new PredicateBuilder<int>().HandleResult(int.IsNegative),
    OnHedging = async args => await NotifyAsync(args.AttemptNumber)
  })
```

Specify concurrent retries + wait until the first successful response

```
new ResiliencePipelineBuilder<HttpResponseMessage>()
  .AddHedging(new HedgingStrategyOptions<HttpResponseMessage>()
  {
    Delay = TimeSpan.Zero,
    ShouldHandle = new PredicateBuilder<HttpResponseMessage>()
      .HandleResult(r => !r.IsSuccessStatusCode)
  })
```

Specify sequential retries + provide a fallback if all attempts failed

```
new ResiliencePipelineBuilder<int>()
  .AddHedging(new HedgingStrategyOptions<int>()
  {
    Delay = TimeSpan.FromSeconds(-1),
    MaxHedgedAttempts = MaxRetries,
    ShouldHandle = new PredicateBuilder<int>().HandleResult(int.IsPositive),
    ActionGenerator = static args => args.AttemptNumber != MaxRetries
      ? () => args.Callback(args.ActionContext) //original action
      : () => Outcome.FromResultAsValueTask(-1) //fallback action
  })
```