FutureProvider + AsyncValue

StreamProvider + AsyncValue

AsyncNotifierProvider + AsyncValue

```
final asyncValue = ref.watch(asyncValueProvider);


return asyncValue.when(
  data: (data) => ShowData(data),
  error: (error, stackTrace) => ShowError(error, stackTrace),
  loading: () => const CircularProgressIndicator(),
);
```

error ▶ dialog

previous data ▶ main UI

opaque loading indicator ▶ semi-transparent loading indicator

previous data ▶ main UI

FutureProvider + AsyncValue

StreamProvider + AsyncValue

AsyncNotifierProvider + AsyncValue

```
@sealed
@immutable
abstract class AsyncValue<T> {
  const AsyncValue._();

  const factory AsyncValue.data(T value) = AsyncData<T>;

  const factory AsyncValue.loading() = AsyncLoading<T>;

  const factory AsyncValue.error(Object error, StackTrace stackTrace) =
      AsyncError<T>;

  ...
}
```

in Riverpod 3.0, AsyncValue will be true sealed class

```
extension AsyncValueX<T> on AsyncValue<T> {
  ...
}
```

value
(AsyncValue)

	previous value (x)	previous value (o)
AsyncLoading	null	previous value
AsyncData	current value	current value
AsyncError	rethrow error	previous value

error, stackTrace
(AsyncValue)

	previous error (x)	previous error (o)
AsyncLoading	null	previous error
AsyncData	null	null
AsyncError	current error	current error

	isLoading **(AsyncValue)**	**hasValue** **(AsyncValue)**	**hasError** **(AsyncValueX)**	
	previous value (x) previous error (x)	previous value (o) previous error (x)	previous value (x) previous error (o)	previous value (o) previous error (o)
AsyncLoading	isLoading (o) hasValue (x) hasError (x)	isLoading (o) hasValue (o) hasError (x)	isLoading (o) hasValue (x) hasError (o)	isLoading (o) hasValue (o) hasError (o)
AsyncData	isLoading (x) hasValue (o) hasError (x)	isLoading (x) hasValue (o) hasError (x)	isLoading (x) hasValue (o) hasError (x)	isLoading (x) hasValue (o) hasError (x)
AsyncError	isLoading (x) hasValue (x) hasError (o)	isLoading (x) hasValue (o) hasError (o)	isLoading (x) hasValue (x) hasError (o)	isLoading (x) hasValue (o) hasError (o)

```
bool get isReloading => (hasValue || hasError) && this is AsyncLoading
```

```
bool get isRefreshing
      => isLoading && (hasValue || hasError) && this is! AsyncLoading
```

```
T? get valueOrNull {
  if (hasValue) return value;
  return null;
}
```

```
T get requireValue {
  if (hasValue) return value as T;
  if (hasError) {
    throwErrorWithCombinedStackTrace(error!, stackTrace!);
  }
  throw StateError(
    'Tried to call `requireValue` on an `AsyncValue` that has no value: $this';
  );
}
```