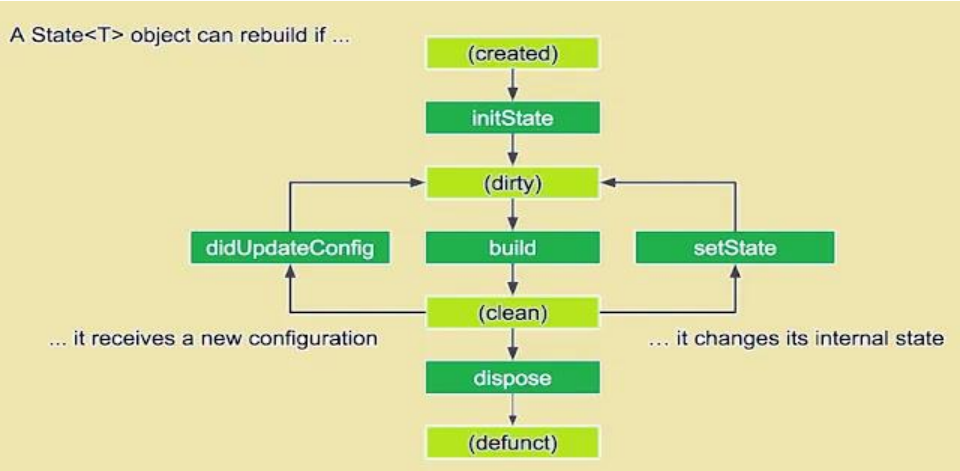
Stateful widget Life Cycle

In Flutter, a **StatefulWidget** manages state that can change over time. It consists of two classes: the **StatefulWidget** itself and the corresponding **State** class that holds the mutable state. Here's the lifecycle of a **StatefulWidget** and its associated **State** class:



1. createState():

- Called when the StatefulWidget is first created.
- Responsible for creating the State object that holds the widget's mutable state.

Example:
Dart

```
@override
State<HomePage> createState() {
  print('create state');
  return _HomePageState();
}
```

2. mounted:

- A boolean property that indicates whether the State object is currently in the widget tree.
- Becomes true after initState() and false before dispose().
- Useful for determining if operations like timers or network requests should proceed.

Dart:

```
_HomePageState() {
  print('constructor, mounted: $mounted');
}
```

3. initState():

- Called once after the State object is created, before the first build.
- Used for:
- Initializing state variables.
- Setting up asynchronous tasks (e.g., fetching data).
- In initState we can’t do two things(await &setState).
- Subscribing to streams or listeners.

Example:

Dart

```
@override
void initState() {
  super.initState();
  print('initState, mounted: $mounted');
}
```

4. didChangeDependencies():

- Called immediately after initState() on the first build, and also whenever the State object's dependencies change.
- Used for:
- Accessing inherited widgets (via BuildContext.inheritFromWidgetOfExactType).
- Responding to changes in inherited data.

Example:

Dart:

```
@override
void didChangeDependencies() {
  super.didChangeDependencies();
  print('didChangeDependencies, mounted: $mounted');
}
```

5. setState():

- Not a lifecycle method, but essential for updating state and triggering rebuilds.

- Used to notify the framework that the State object's data has changed, causing the build() method to be called again.
- Example:

Dart

```
@override
void setState(VoidCallback fn) {
  print('setState');
  super.setState(fn);
}

void _incrementCounter() {
  setState(() {
    _counter++;
  });
}
```

6. build():

- Called whenever the widget needs to be redrawn.
- Responsible for building the widget's UI based on the current state.
- Example:

Dart

```
@override
Widget build(BuildContext context) {
  print('build method');
```

7. deactivate():

- Called when the widget is removed from the widget tree, but might be inserted back later.
- Used for cleaning up resources that should not be kept alive when the widget is inactive.

Example

Dart:

```
@override
void deactivate() {
  super.deactivate();
  print('deactivate, mounted: $mounted');
}
```

8. dispose():

- Called when the State object is permanently removed from the widget tree.
- Used for:
- Canceling timers or subscriptions.
- Cleaning up resources that are no longer needed.

Example
Dart:

```
@override
void dispose() {
  super.dispose();
  print('dispose, mounted: $mounted');
}
```

9. **reassemble:**
Called when the application is reassembled during hot reload.

Example
Dart:

```
@override
void reassemble() {
  super.reassemble();
  print('reassemble, mounted: $mounted');
}
```

Remember:

- Stateful widgets are ideal for UI elements that change dynamically in response to user interactions or data updates.
- Understanding the lifecycle methods is crucial /important for managing state effectively and building responsive Flutter applications