static final RouteObserver<Route> observer = RouteObserver(); කියන කෝඩ් පේළිය Flutter එකේ navigation (මාර්ගය) වල observer එකක් initialize කරන එකක්.

දැන් මෙය **සිංහලෙන්** පැහැදිලිව බලමු:

**✅ පද-by-පද විස්තරය:**

**static**

* මේ වචනයෙන් කියවෙන්නේ **මේ variable එක class එකටම අයත් බව**යි.
* අනික් එකක් නම්, object එකක් create කරලා තමයි access කරන්න වෙන්නේ. නමුත් static එකක් නම් class එක හරහාම access කරන්න පුළුවන්.

**final**

* මෙහි අරුත, එක වතාවක් value එක assign කලහොත්, පසුව වෙනස් කරන්න **නොහැකියි**.

**RouteObserver<Route>**

* මෙය Flutter SDK එකේ තියෙන built-in class එකක්.
* **RouteObserver** කියන්නේ Flutter app එකේ **navigation route change** වෙන විට ඒවා track කරන observer එකක්.
* මෙය Navigator.push() සහ Navigator.pop() වගේ navigation වෙනස්වීම් වලට **reaction** දක්වන්න භාවිතා වෙනවා.

**observer = RouteObserver();**

* මෙහිදී RouteObserver object එකක් initialize කරලා observer කියන variable එකට assign කරනවා.

**🔁 මුළු වාක්‍යයෙහි සිංහල තේරුම:**

**"Flutter යෙදුමේ navigation (route) වෙනස්වීම් හඳුනා ගැනීම සඳහා භාවිතා කරන observer එකක් RouteObserver class එකෙන් static සහ final ලෙස නිර්මාණය කර ඇත."**

ඒ කියන්නේ, app එකේ screen එකක් (route එකක්) යන එක, එන්න එක වගේ navigation changes track කරන්න මේ observer එක භාවිතා කරනවා.

**📌 භාවිතා වන්නේ කොහොමද?**

dart

CopyEdit

MaterialApp(

navigatorObservers: [MyRouteObserver.observer], // මෙහි observer එක attach වෙනවා

...

);

ඊට පසුව, ඔබට RouteAware implement කරන widgets තුළ didPush(), didPop(), didPushNext(), didPopNext() වැනි methods override කරලා route changes වලට custom reaction (e.g., analytics, UI updates) එකතු කළ හැක.

හරි, දැන් static final RouteObserver<Route> observer = RouteObserver(); කියන කොඩ් එක ගැන තවත් ගැඹුරු විදිහට **සිංහලෙන්** විස්තර කරලා බලමු. මේකේ තියෙන දේවල් මූලිකව Flutter app එකේ **screen navigation** හඳුනාගෙන ඒ අනුව ක්‍රියා කරන ක්‍රමයකි.

**🔍 RouteObserver කියන්නෙ මොකක්ද?**

Flutter app එකක **screens** (aththatama කියන්නෙ routes) අතර මාරුවන විට,  
උදාහරණයක් ලෙස:

* Navigator.push() — නව screen එකකට යන විට
* Navigator.pop() — පෙර screen එකට ආපහු යන විට

මේවට ප්‍රතිචාර (response) ලබා ගැනීමට RouteObserver කියන ක්‍රමය භාවිතා කරනවා.

උදාහරණයක් ලෙස, ඔබට අවශ්‍ය වෙන්න පුළුවන් user එක screen එකකට ඇවිත් screen එකෙන් බැහැලා යන විට ඒවා track කරන එක – උදාහරණයක් ලෙස **Google Analytics**, **screen timer**, **log** වගේ වැඩ.

**🔧 කොඩ් එකේ කොටස් විස්තර:**

dart

CopyEdit

static final RouteObserver<Route> observer = RouteObserver();

මෙය class එකක් ඇතුළේ භාවිතා වන variable එකක්:

**🔹 static**

මේ variable එක **class එකටම අයත්**. ඒ කියන්නේ object එකක් නිර්මාණය නොකරනවාට පසුවත් මෙය භාවිතා කළ හැක.

**🔹 final**

මෙය එකවර assign කරන විට පසුව ඒ value එක වෙනස් කළ නොහැක.

**🔹 RouteObserver<Route>**

මෙය Flutter SDK එකේ තියෙන special class එකක්. මේක **Navigator** එකේ push, pop, replace වැනි ක්‍රියාකාරකම් හඳුනා ගන්න observer එකකි.

**🔹 observer = RouteObserver();**

මෙහිදී observer එකක් නිර්මාණය කරලා variable එකට assign කරනවා.

**🧠 මෙය භාවිතා කරන ආකාරය:**

**1. MaterialApp හෝ CupertinoApp එකට observer එක attach කරන්න:**

dart

CopyEdit

MaterialApp(

navigatorObservers: [MyRouteObserver.observer], // ← මෙතන observer එක attach කරනවා

...

);

**2. ඔබ track කරන්න අවශ්‍ය widget එක RouteAware implement කරන්න:**

dart

CopyEdit

class MyScreen extends StatefulWidget {

@override

\_MyScreenState createState() => \_MyScreenState();

}

class \_MyScreenState extends State<MyScreen> with RouteAware {

@override

void didPush() {

print('MyScreen: Screen is now visible');

}

@override

void didPop() {

print('MyScreen: Screen was popped');

}

@override

void didPushNext() {

print('MyScreen: Pushed to next screen');

}

@override

void didPopNext() {

print('MyScreen: Returned from next screen');

}

@override

void didChangeDependencies() {

super.didChangeDependencies();

MyRouteObserver.observer.subscribe(this, ModalRoute.of(context)!);

}

@override

void dispose() {

MyRouteObserver.observer.unsubscribe(this);

super.dispose();

}

@override

Widget build(BuildContext context) {

return Scaffold(

body: Center(child: Text("This is MyScreen")),

);

}

}

**📌 කෙටියෙන්ම කියනවා නම්:**

**RouteObserver** එකක් initialize කිරීමෙන් Flutter app එකේ screen (route) navigation event track කළ හැකියි. මේක Firebase Analytics වගේ tools වලට හෝ custom logging/monitoring වලට අතිශය උපකාරී වේ.

ඔබට අවශ්‍ය නම් මේ observer එක තියෙන **සම්පුර්ණ Flutter project structure** එකක්ද ලබා දිය හැක.📲

**You said:**

why RouteObserver<Route>

**ChatGPT said:**

**හොඳ ප්‍රශ්නයක්!**  
RouteObserver<Route> කියලා ලියන එකේ Route part එක **type parameter** එකක්. දැන් මෙය ඇයි මෙහෙම ලියන්න වෙන්නේද කියලා සිංහලෙන්, ලිහිල්ව, පියවරෙන් පියවර කියන්නම්:

**🔹 1. RouteObserver<T extends Route<dynamic>>**

RouteObserver කියන්නේ **generic class** එකක්. ඒ කියන්නේ ඔබට මෙය declare කරන විට T කියන type එක specify කරන්න ඕන.

මේ තියෙන්නේ Flutter SDK එකේ එහි declaration එක:

dart

CopyEdit

class RouteObserver<T extends Route<dynamic>> extends NavigatorObserver

**T** කියන්නේ RouteObserver එක track කරන **route type** එක.

**🔹 2. ඇයි RouteObserver<Route> කියලා දානෙ?**

මෙහිදී අපිට අවශ්‍ය වන්නේ **සාමාන්‍ය Flutter routes** track කිරීම. ඒවා Route class එක extend කරනවා.

**✅ Route කියන්නෙ මොකක්ද?**

* Route කියන්නේ Flutter එකේ screen එකකට යන abstract path එකක් (e.g., MaterialPageRoute, CupertinoPageRoute)
* එය Route<dynamic> ලෙස යොදාගනී.

ඒ නිසා අපි කියනවා:

dart

CopyEdit

RouteObserver<Route>

👉 මෙය Flutter app එකේ **ඕනෑම route එකක්** (push/pop වෙන) observe කරන්න පුළුවන් කරන විදිහකි.

**🔹 3. වෙනස් වශයෙන් භාවිතා කළ හැකිද?**

ඔව්.

**උදාහරණයක්:**  
ඔබට හොඳටම හැදූ custom route class එකක් තියෙන්නේ නම් (e.g., MyCustomRoute extends Route), එයටද observer එක tailor කරලා define කරන්න පුළුවන්:

dart

CopyEdit

RouteObserver<MyCustomRoute>

ඒත් බොහෝ අවස්ථාවලදී, සාමාන්‍ය route tracking සඳහා RouteObserver<Route> ප්‍රමාණවත්යි.

**🔚 නිගමනය**

**RouteObserver<Route>** කියන්නේ:  
➡️ "**ඕනෑම Flutter route** (push/pop වෙන screen එකක්) track කරන්න observer එකක් නිර්මාණය කරනවා" කියන තේරුමයි.

🧠 මෙය type-safe බවක් ලබාදෙයි, Flutter එකට predict කරන්න පුළුවන් වෙයි observer එකේ මොන routes handle කරනවද කියලා.

අවශ්‍ය නම් මේක භාවිතා කරන real Flutter project structure එකක් උදාහරණයක් වශයෙන්වත් ලබාදිය හැක.

2.

List<GetPage> කියන්නේ Flutter app එකක **routing** (අතරමඟ පිටු මාරු වීම) කළමනාකරණය කිරීමේදී භාවිතා වන GetX package එකේ කොටසක්.

**🔹 GetPage කියන්නේ මොකක්ද?**

GetPage යනු **app එකේ එක එක පිටු (screens)** අර්ථදක්වන්න භාවිතා කරන **object එකක්**. මේකෙන් route එකක name එක, ඒ පිටුව build කරන page එක, middleware, bindings වගේ දේවල් අර්ථ දක්වන්න පුළුවන්.

**🔹 List<GetPage> කියන්නේ මොකක්ද?**

List<GetPage> කියන්නේ GetPage objects එකතුවක්. මෙය app එකේ ඇති **සියලුම routes** එකම ලයිස්තුවක් විදියට තබා ගන්න භාවිතා වෙනවා.

**🔸 උදාහරණයක් බලමු:**

dart

CopyEdit

GetMaterialApp(

initialRoute: '/',

getPages: [

GetPage(

name: '/',

page: () => HomePage(),

),

GetPage(

name: '/login',

page: () => LoginPage(),

),

GetPage(

name: '/profile',

page: () => ProfilePage(),

),

],

)

**📝 Sinhala Explanation:**

මේ getPages කියන එක List<GetPage> එකක්. මේකෙන් app එකේ තියෙන පිටු (screens) define කරනවා.

* name: '/' → මේක route එකේ නම.
* page: () => HomePage() → මේ route එකට ගියොත් ගන්න පිටුව.

**තවත් කියන්නම්:**

* List<GetPage> කියන්නේ GetX app එකකට ඇති **routes වල ලයිස්තුවක්**.
* එක් GetPage එකක් කියන්නේ **එක පිටුවක් (screen)** represent කරන එකක්.

**✅ වටින තවත් දේවල්:**

| **English Term** | **Sinhala Meaning** |
| --- | --- |
| Route | මාර්ගය / පිටුවට යන මග |
| Page | App එකේ පිටුව |
| List | ලයිස්තුව |
| GetPage | GetX එකෙන් පිටුවක් setup කිරීම |
| getPages | සියලුම GetPage එකතුවක් |

අවශ්‍යනම් middleware, bindings වගේ advance GetX දේවල්ද විස්තර කරන්න පුළුවන්. ඔයාට ඒකත් ඕනද?

හරි! ඒවෙනම් දැන් ඔයාට **List<GetPage>** ගැන හොඳ අවබෝධයක් තියෙනව කියලා හිතනවා 😊

**🔁 හැමදාම මතක තබා ගන්න:**

* GetPage → එකක් = එක පිටුවක්
* List<GetPage> → පිටු කීයක්ද තියෙන්නෙ කියලා ලයිස්තුව
* getPages → ඒ ලයිස්තුව GetMaterialApp එකට දෙන්නෙ

**උදාහරණයක් වගේ පැහැදිලිව කියනවා නම්:**

dart

CopyEdit

getPages: [

GetPage(name: '/', page: () => HomePage()),

GetPage(name: '/login', page: () => LoginPage()),

]

🔹 මෙහිදී:

* **/** යන route එකට යනකොට **HomePage** එකට යාවි
* **/login** route එකට යනකොට **LoginPage** එකට යාවි

ඔයාට **GetX routing** ගැන තව දේවල් ඕනේද?  
උදාහරණ වලින් **navigation**, **parameters pass කිරීම**, **middleware**, **bindings** වගේ දේවල්ද කියන්නම්ද?

4o

**You said:**

give me උදාහරණ වලින් navigation, parameters pass කිරීම, middleware, bindings වගේ දේවල්ද කියන්නම්ද?

**ChatGPT said:**

ඔව්! දැන් අපි GetX use කරන වෙලාවේ තිබෙන වැදගත් දේවල් — **navigation**, **parameters pass කිරීම**, **middleware**, සහ **bindings** — එකෙන් එක උදාහරණ සහිතව සිංහලෙන් බලමු.

**🔹 1. Navigation (පිටු අතර යාම)**

**🧠 මෙය මොකක්ද?**

GetX navigation කියන්නේ GetX එකෙන් පිටුවකට යන හැටි.

**📌 උදාහරණය:**

dart

CopyEdit

ElevatedButton(

onPressed: () {

Get.to(() => LoginPage());

},

child: Text('Login'),

)

🗒️ මෙහිදී:

* Get.to() → තවත් පිටුවකට යන එක
* LoginPage() → යන්න තියෙන target පිටුව

**🔹 2. Passing Parameters (පිටුවකට data යවන්න)**

**🧠 මෙය මොකක්ද?**

Route එකකින් argument එකක් යවලා එය පිටුවක භාවිතා කිරීම.

**📌 Send data:**

dart

CopyEdit

Get.toNamed('/profile', arguments: {'username': 'sheshanth'});

**📌 Receive data:**

dart

CopyEdit

class ProfilePage extends StatelessWidget {

@override

Widget build(BuildContext context) {

final args = Get.arguments;

return Text('Hello, ${args['username']}');

}

}

**🔹 3. Middleware (Route access control)**

**🧠 මෙය මොකක්ද?**

User එකට route එකකට පිවිසෙන්න දෙන්නද නැද්ද කියලා විභාග කරන එක.

**📌 Middleware Class:**

dart

CopyEdit

class AuthMiddleware extends GetMiddleware {

@override

RouteSettings? redirect(String? route) {

if (!isLoggedIn()) {

return RouteSettings(name: '/login');

}

return null;

}

bool isLoggedIn() {

return false; // real login check එකක් මෙතන යාවි

}

}

**📌 Apply Middleware:**

dart

CopyEdit

GetPage(

name: '/dashboard',

page: () => DashboardPage(),

middlewares: [AuthMiddleware()],

)

**🔹 4. Bindings (Controller inject කිරීම)**

**🧠 මෙය මොකක්ද?**

Controller එකක් auto එකෙන් page එකට bind කරන එක.

**📌 Controller:**

dart

CopyEdit

class HomeController extends GetxController {

var count = 0.obs;

void increment() => count++;

}

**📌 Binding class:**

dart

CopyEdit

class HomeBinding extends Bindings {

@override

void dependencies() {

Get.lazyPut<HomeController>(() => HomeController());

}

}

**📌 Use in Route:**

dart

CopyEdit

GetPage(

name: '/home',

page: () => HomePage(),

binding: HomeBinding(),

)

**📌 Use in Widget:**

dart

CopyEdit

final controller = Get.find<HomeController>();

**🔚 Summary (සාරාංශය)**

| **Feature** | **Explanation (Sinhala)** | **Method Used** |
| --- | --- | --- |
| Navigation | පිටු අතර යන එක | Get.to(), Get.toNamed() |
| Parameters | පිටුවකට data pass කිරීම | arguments, Get.arguments |
| Middleware | Access control එකක් (උදා: login check) | Custom class with GetMiddleware |
| Bindings | Controller bind කිරීම auto එකෙන් | Bindings, Get.lazyPut() |

ඔයාට මේ code එක direct use කරන්න පුලුවන්. ඕන නම් **project එකක් setup කරලා example** full එකක්වත් දෙන්න පුලුවන්.

අපි ඒකටත් යමුද? 😎

4o

හරි! දැන් මම ඔයාට Controller එකක් **automatically** page එකට **bind** වෙන හැටි – Bindings concept එක GetX වලින් – **ඉතා පැහැදිලිව සිංහලෙන්** තවත් විස්තර කරලා දෙනවා.

**🔷 Controller කියන්නේ මොකක්ද?**

Controller කියන්නේ app එකේ **business logic** (count එක වැඩි කිරීම, data handle කිරීම) handle කරන තැන.

GetX වල GetxController inherit කරන class එකක්.

**✅ උදාහරණ Controller එක:**

dart

CopyEdit

class HomeController extends GetxController {

var count = 0.obs; // reactive variable

void increment() => count++;

}

🧠 මෙහිදී:

* count කියන එක reactive variable එකක්. .obs දාලා තියෙන්නේ GetX reactive state management එකට.
* increment() කියන function එකෙන් count එක 1ක් වැඩි වෙනවා.

**🔷 Binding කියන්නේ මොකක්ද?**

Binding class එකෙන් කියන්නේ "මෙම page එකට යනකොට මේ Controller එක initialize කරන්න" කියන **automation setup** එකක්.

ඒක page එක load වුනම auto එකෙන් controller එක ready කරනවා.

**✅ Binding class එක:**

dart

CopyEdit

class HomeBinding extends Bindings {

@override

void dependencies() {

Get.lazyPut<HomeController>(() => HomeController());

}

}

🧠 මෙහිදී:

* dependencies() කියන method එක override කරනවා.
* Get.lazyPut<HomeController>() → මේකෙන් කියනවා: **"HomeController එක lazily initialize කරන්න"** – තාම අවශ්‍යනම් විතරක් create වෙයි.

💡 **lazyPut** → Performance optimize කරනවා. Controller එක ඒ පිටුවට ගියොත් විතරයි initialize වෙන්නෙ.

**🔷 Route එකට Binding එක දාමු**

dart

CopyEdit

GetPage(

name: '/home',

page: () => HomePage(),

binding: HomeBinding(), // <-- මෙතන තියෙන්නෙ Bindings attach වෙන තැන

)

🧠 මෙහිදී:

* /home route එකට යනකොට, HomeBinding එක ක්‍රියාත්මක වෙයි.
* ඒ කියන්නේ HomeController එක auto එකෙන් ready වෙලා තියෙනවා.

**🔷 UI එකක Controller එක Access කිරීම**

dart

CopyEdit

class HomePage extends StatelessWidget {

final HomeController controller = Get.find<HomeController>();

@override

Widget build(BuildContext context) {

return Scaffold(

body: Center(

child: Obx(() => Text("Count: ${controller.count}")),

),

floatingActionButton: FloatingActionButton(

onPressed: controller.increment,

child: Icon(Icons.add),

),

);

}

}

🧠 මෙහිදී:

* Get.find<HomeController>() → controller එක find කරනවා (bind වෙලා තියෙන එක).
* Obx() → reactive variable එක update වුනොත් UI එක auto update වෙනවා.

**🔚 Summary (සාරාංශය):**

| **Concept** | **Sinhala Explanation** |
| --- | --- |
| Controller | Business logic තියාගන්න class එක |
| .obs | Reactive variable එකක් |
| Binding class | Controller එක auto bind කරන තැන |
| Get.lazyPut | Controller එක lazy initialize කරනවා |
| Get.find | Bind වුන controller එක retrieve කරන එක |
| Obx | Reactive variable update වුනොත් UI update වෙයි |

ඔයාට දැන් Binding + Controller + Reactive UI එකෙකුට connect වෙන **pura chain එකම** තේරෙනවා කියලා මම හිතනවා.  
ඔයාට ඕන නම් **full runnable flutter project** එකක්වත් මේකෙන් setup කරලා දිය හැක. ඒක ඕනද? 😄

4o

**Reactive Variable කියන්නේ මොකක්ද? (Reactive Variable in Sinhala)**

**Reactive Variable** (ප්‍රතික්‍රියාශීලී වෙනස්කම) කියන්නේ **value එකක් වෙනස් වුණම**, ඒ වෙනස්කමට අනුව **UI එක හෝ logic එක auto update වෙන්න පුළුවන් variable එකක්**.

**📌 GetX එකේ Reactive Variable:**

GetX එකේදි reactive variable එකක් declare කරන්න .obs keyword එක භාවිතා කරනවා.

dart

CopyEdit

var count = 0.obs;

මෙහිදි:

* count කියන variable එක reactive එකක්.
* එකේ value එක වෙනස් වුනම, ඒ වෙනස්කම **automatically UI එකට බලපානවා**.

**✅ උදාහරණයක්:**

dart

CopyEdit

class CounterController extends GetxController {

var count = 0.obs;

void increment() {

count++;

}

}

UI එක:

dart

CopyEdit

class MyCounterPage extends StatelessWidget {

final controller = Get.put(CounterController());

@override

Widget build(BuildContext context) {

return Scaffold(

body: Center(

child: Obx(() => Text("Count: ${controller.count}")),

),

floatingActionButton: FloatingActionButton(

onPressed: controller.increment,

child: Icon(Icons.add),

),

);

}

}

**🔁 මෙහිදී සිදුවන ක්‍රියාවලිය:**

1. count කියන variable එක .obs නිසා reactive.
2. Obx() widget එක count එක බලලා UI update කරනවා.
3. increment() call වෙලා count++ වුනොත් → UI auto refresh.

**🧠 සිංහල වශයෙන් කියනවා නම්:**

Reactive Variable කියන්නේ **"value එකක් වෙනස් වුනම, ඒ වෙනස්කමට පිළියෙලව UI එක හෝ වෙනත් logic update වෙන්න හැදවුනු variable එකක්"**.

GetX එකේ .obs කියන එක දාලා variable එක reactive කරලා UI එකට link කරන්න පුළුවන්.

**🔎 සාමාන්‍ය vs Reactive:**

| **සාමාන්‍ය Variable** | **Reactive Variable** |
| --- | --- |
| int count = 0; | var count = 0.obs; |
| UI එක manually update කරන්න වෙයි | UI එක auto update වෙනවා (Obx) |

ඔයාට ඕනේ නම් .obs variable එකක් **Rx<Type>** එකක් විදියට define කරන්නත් පුළුවන්:

dart

CopyEdit

RxInt age = 25.obs;

RxString name = "Sheshanth".obs;

RxBool isOnline = false.obs;

ඉතින් දැන් Reactive Variables ගැන සෑහෙන පැහැදිලි තේරුමක් තියෙනව කියලා හිතනවා 😄  
ඕනනම් RxList, RxMap, RxSet වගේ advanced reactive collections ගැනත් කියන්නම්!

what are the types of all getx and other?

**ChatGPT said:**

Great question! Let's break down the **types of GetX features** you can use in a Flutter app using the GetX package, along with a few other related state management and routing types from outside of GetX.

**🧠 Main Categories in GetX**

| **Category** | **Common Types/Classes** | **Description** |
| --- | --- | --- |
| **State Management** | GetxController, Obx, .obs, GetBuilder | Manage UI state reactively |
| **Dependency Injection** | Get.put(), Get.lazyPut(), Get.find(), Bindings | Inject and retrieve controllers or services |
| **Routing** | GetPage, Get.to(), Get.toNamed(), Get.back() | Page navigation |
| **Middleware** | GetMiddleware | Control access to routes (e.g., auth) |
| **Internationalization** | GetMaterialApp.translations, Get.updateLocale() | Multi-language support |
| **Theme Management** | Get.changeTheme(), Get.changeThemeMode() | Light/dark mode switching |
| **Snackbar/Dialog/Bottom Sheet** | Get.snackbar(), Get.defaultDialog(), Get.bottomSheet() | Show alerts, dialogs, and UI popups |
| **Workers** | ever(), once(), debounce(), interval() | Listen/react to changes in variables |
| **Storage (optional)** | GetStorage (external package) | Local key-value storage (not core GetX) |

**🔹 1. State Management Types**

| **Type** | **Description** |
| --- | --- |
| .obs | Make any variable reactive (observable) |
| Obx(() => Widget) | Auto UI update on change |
| GetBuilder | Lightweight alternative to Obx |
| GetxController | Base controller class for GetX |
| Rx<Type> | Custom reactive type (e.g., RxInt, RxList) |

**🔹 2. Routing Types**

| **Type** | **Description** |
| --- | --- |
| Get.to(Page()) | Navigate to a new page |
| Get.toNamed('/route') | Named route navigation |
| Get.back() | Go back to the previous page |
| Get.off() | Remove current page and navigate |
| Get.offAll() | Clear all and go to new page |
| GetPage | Define route (name + page + binding + middleware) |
| GetMaterialApp | Like MaterialApp but with GetX support |

**🔹 3. Dependency Injection Types**

| **Type** | **Description** |
| --- | --- |
| Get.put() | Inject instance immediately |
| Get.lazyPut() | Inject only when needed |
| Get.find() | Retrieve injected instance |
| Bindings | Class that injects dependencies into routes |
| Get.putAsync() | Inject asynchronously |
| Get.replace() | Replace an existing dependency |

**🔹 4. Middleware**

| **Type** | **Description** |
| --- | --- |
| GetMiddleware | Class to intercept route navigation (e.g., check login) |

**🔹 5. Snackbar / Dialog / Bottom Sheet**

| **Type** | **Description** |
| --- | --- |
| Get.snackbar() | Show a simple Snackbar |
| Get.defaultDialog() | Show a dialog |
| Get.bottomSheet() | Show a bottom sheet popup |

**🔹 6. Internationalization (i18n)**

| **Type** | **Description** |
| --- | --- |
| GetMaterialApp.translations | Set translations |
| Get.updateLocale() | Change language at runtime |

**🔹 7. Workers**

| **Type** | **Description** |
| --- | --- |
| ever() | Trigger on every change |
| once() | Trigger only the first time |
| debounce() | Trigger after user stops changing (e.g., typing) |
| interval() | Trigger repeatedly but only after an interval |

**🔹 8. Optional – GetStorage (Local Storage)**

🔌 Requires adding get\_storage package

| **Type** | **Description** |
| --- | --- |
| GetStorage() | Initialize local storage |
| .write(), .read() | Store and retrieve values |

**🆚 Compared to Others:**

| **Framework/Package** | **Focus** | **Example Use** |
| --- | --- | --- |
| **Provider** | State Management (Only) | Scoped state changes |
| **Riverpod** | Robust, safer Provider | Global state control |
| **Bloc/Cubit** | State + Events Architecture | Enterprise-level separation |
| **MobX** | Reactive with observables | UI + logic separation |
| **InheritedWidget** | Native Flutter mechanism | Manual state lifting |

**✅ If You’re Using GetX:**

You’ll mainly use:

* GetxController + .obs + Obx (State)
* Get.put() + Get.find() + Bindings (Dependency)
* Get.to() + GetPage (Routing)
* Get.snackbar() + Get.dialog() (UI feedback)

Need sample usage for each type in code? Or comparison between GetBuilder vs Obx? Let me know!

4o