**GERENCIAMENTO DE ESTADO**

* Vamos usar o provider, multprovider como gerenciador
* **Dependências** 
  + provider
  + intl
* **Inherited Widget**: é um widget que vai nos ajudar a trafegar informações na arvore de componentes da aplicação.
* **O Problema**: quando queremos passar dados de um widigt que esta´localizado longe na árvore. Temos a forma direta quando o pai mandar parâmetros para o filho definido no construtor, temos a indireta passos via construtor uma função .
* **Solução**: colocar o **inherited widget** dentro da árvore de componente e a partir dele conseguimos ter uma comunicao direta com vários componentes . o provider vai se basear neste conceito.
* **O que é estado**?
  + dados que afeta a interface do user
  + **tipos**
    - **Estado da aplicação**
    - **Estado do widget(LOCAL)**
* **Como o PROVIDER funciona**
  + É necessário injetar o provider em algum ponto da arvore, não necessáramente a raiz.
    - Uma vez colocado o provider(provedor) que dentro dele tem um estado que ele esta gerenciando ou um conjunto de informações, ele será um container, e os filhos poderão acessar o providder diretamente sem que vc precise passar essas informações de um componente para outro.
  + Acesso: of(contexto)
  + A comunicação dos buildcontexts ocorrem de cima para baixo
* **Mixins**
  + Entenda primeiramente q dart não aceita multi-herança
  + Aceita mais de um mixins
  + O mixin é como se ele desse control c no bloco de código mixin e desse control v na class
  + É como uma mistura. Pega um trecho de código e joga
* **Provider:**
  + É algo que envolve (wrapper) o inheritedWidget e como isso vai facilitar o uso deste mecanismo e na reutilização. Ele trabalha muito com o C**hangeNotifier** que nada mais é q o padrão observer que é uma parte essencial nesta questão de reatividade
  + **Padrão observer:** se basea em vc se registrar e assim q houver mudança vc será notificado
  + **notifyListeners:** método da classe ChangeNotifier que vai notificar sempre q houver mudanças. É parte essencial na programação reativa,
  + **ChangeNotifierProvider:**
    - Create – para criar o provider
  + **Provider.of<T> (contexto)** para pegar o provider
    - **Listen** segundo parâmetro q indica se vc está ou não interessado nas mudanças. Isso será útil para quando vc estiver criando uma parte da interface gráfica que apenas lidar com partes que são imutavei, p ex o titulo do produto, descricao
* **Formas de Usar o Provider**
  + **Do – create:**
  + **Usando valor existente:**
    - .value
    - Usamos isso quando tivermos um ChangeNotifier criado anteriormente
* **Consumer<T>:** alternativa do provider.of. como o consumer é um componente, vc pode colocar especificamente no componente que vc sabe que vai haver mudanças
  + **Builder:** onde as coisas serão modificadas
    - Contexto
    - Value
    - Child – recebe o child se o consumer tiver
  + **Child**
* **MultiProvider**
  + Vai pegar vários providers