

week4

Layout → flutter에서 화면 배치 관련한 위젯 →

Container class

- 컨테이너 위젯은 child가 없을 경우 페이지 내에서 할 수 있는 한 최대한의 공간을 차지하려 함.
- 만약 child 위젯이 있다면? Container 위젯은 child 위젯의 크기로 줄어들어 버림.

SafeArea 위젯

- 개발자가 보여지기를 원하는 콘텐츠가 화면 밖으로 빠져나가지 않도록 경계 설정하는.
- container에 alt enter 누르고 → wrap with widget → SafeArea
- `margin: EdgeInsets.all(20)`: 화면 가장자리로부터 20 픽셀씩 떨어지도록. 상하좌우를.
- 위아래 간격, 좌우 간격 보다 편하게 지정: `EdgeInsets.symmetric`
- flutter inspector 메뉴 클릭 → 가이드라인 나옴 → 좌우로 몇 픽셀 떨어졌는지
- `padding`: `EdgeInsets.all(20)` → 컨테이너 안의 텍스트 위치 조절

정리하면

- `margin` 컨테이너가 스크린의 가장자리에서 일정 간격 갖도록 할 때 ⇒ 즉 위젯의 바깥쪽
- `padding` 컨테이너가 포함하고 있는 요소가 컨테이너의 가장자리에서 일정 간격 갖도록 할 때 ⇒ 위젯의 안쪽 간격 조절
- container 오직 하나만의 child 가짐

column: child 아닌 children 가짐. 여러개의 위젯들을 세로로 나열하기 위한 위젯이기 때문.

```
child: Column(  
  children: <Widget>[], // 여기 []에 위젯들의 리스트 나열하기  
  
  // 이 안에 세개의 컨테이너 위젯을 만들어보겠음 -> column 때문에 세개의 위젯은 별다른 설정 없이도 세로로 정렬 됨.
```

- 세로축 방향으로 가능한 모든 공간을 다 차지함. 그러나 가로축으로는 children의 가로방향으로 제한됨. → column 위젯을 center widget으로 감싸줌 (wrap with center) → column 위젯이 center 위젯의 child로 오면, column 위젯이 세로 축의 모든 가능한 공간을 차지하기 때문에 center 위젯은 세로축 통제권을 잃고, column 위젯의 children의 시작점을 기준으로 오직 가로축 정렬에만 관여하게 됨. → center 위젯은 좌우에서만 container 위젯의 위치 관여

⇒ container 정중앙에 배치하기 위해 children의 세로축 위치 조절해주는 main.axisargument를 불러와서 center 속성을 줘야함. `mainAxisAlignment: MainAxisAlignment.center`

→ column 위젯이 세로축의 모든 공간 차지하지 못하게 하려면? `mainAxisSize: MainAxisSize.min`

⇒ 컨테이너의 필수 공간만큼으로 세로 영역이 줄어들어, center 위젯 상하의 통제권 확보하게 됨.

⇒ center 위젯 상하좌우에서 container의 위치 통제하게 됨.

- 3개의 container 정렬 순서 바꾸기

`verticalDirection: VerticalDirection.up` → 컨테이너 밑에서부터 1,2,3 순서로 쌓아올림

`down`으로 바꾸면 위에서 아래로 정렬

- 같은 간격을 가지고 children 위젯 배치 기능

`mainAxisAlignment: MainAxisAlignment.spaceEvenly` → 3개의 컨테이너가 같은 간격을 가지고 세로로 정렬

`mainAxisAlignment: MainAxisAlignment.spaceBetween` → 컨테이너가 정확히 스크림의 상중하에 위치하게 됨

- column 위젯의 children 가로축 정렬

`crossAxisAlignment: CrossAxisAlignment.end` →

- 컨테이너 세개 모두 한번에 오른쪽 끝으로 정렬

`invisible Container`

- 컨테이너들 가로방향으로 꽉 채우고 싶다면

`crossAxisAlignment: CrossAxisAlignment.stretch`

- 컨테이너들 사이에 간격을 두고 싶다면

(컨테이너 사이에 공간을 집어넣어 간격을 만든다고 생각하면 됨) `SizedBox`

Row widget(가로축 정렬)도 column 위젯의 속성이 그대로 적용됨.


→ 컨테이너 세로 아닌 가로로 정렬됨.

(컨테이너 사이에 간격 줄때 `sizedbox` 사용하는데 `height` 아닌 `width`로 간격 지정해주면 됨.)

★ 레이아웃 확인

Flutter Layout Cheat Sheet

Do you need simple layout samples for Flutter? I present you my set of Flutter layout code snippets. I will keep it short, sweet and simple with loads of visual examples. Still, it is work in progress - the

 <https://medium.com/flutter-community/flutter-layout-cheat-sheet-5363348d037e>



Navigator

1. Route의 개념
2. Navigator의 정의와 `push`, `pop` 함수, `stack` 자료구조
3. `MaterialPageRoute` 위젯과 `context`
4. 페이지 이동 기능 구현 완성

스마트 폰 상에서 보여지는 하나의 페이지를 의미한다고 이해하면 됨. 모든 앱 페이지를 관리하여 `route` 객체를 관리하는게 `navigator`. `navigator` 이해하려면 `stack` 자료 구조 이해해야함.

`stack`?

- 자료가 밑에서부터 차곡차곡 쌓여진다고 보면 됨
- `push` 메서드를 통해서만 데이터를 쌓을 수 있음
- `pop` 메서드를 통해 데이터 지울 수 있음.

⇒ 즉, flutter에서 앱 페이지 `route`에 대한 관리는 `navigator` 위젯이 담당하고. `navigator`는 앱 페이지에 대한 데이터 관리 위해 데이터 차곡차곡 쌓는 방식의 `stack` 자료구조 사용하고. 이 자

로구조에서 데이터 추가할 때는 push 사용, 없앨때는 pop 메서드 사용함.

```
Navigator.push(context, route)
```

// context: context가 가지고 있는 위젯 트리의 위치 정보에 근거해, 화면상에 보여지는 페이지가 어떤 페이지인지 확인하고 이 페이지 위에 push 함수가 이동하길 원하는 새로운 route 쌓아올려야 하기 때문. => 즉, first page 위로 second page 올리려면, first page의 위치 제대로 알아야 하기 때문.

// route 자리에... MaterialPageRoute() 대입 -> 이는 builder를 argument로 가짐. builder는 어떤 위젯이 MaterialPageRoute의 도움을 받아서 생성되어야 할지를 정의함. 우리의 경우에는 second page가 됨.

```
// push와 달리 pop 메서드는 구현 아주 간단함  
Navigator.pop(context)
```

home argument → 앱이 처음 실행됐을 때, 제일 먼저 화면에 보여질 router를 호출하는 것.

```
home: ScreenA()
```

```
// ScreenA 먼저 import해줘야 함
```

```
import 'scr' ~
```

멀티 페이지 이동 기능 구현

→ home 대신 initialRoute argument 사용함. 두개가 동시에 존재하면 에러 발생하니 주의할 것

Map 자료구조

- Key : Value
- (String : Widget builder)

```
initialRoute: '/' // flutter에서는 첫 router의 이름 슬래쉬로 지정해줌.  
routes: {
```

```

    '/' : (context) => ScreenA(), // 이동을 원하는 router 지정
  },

```

```

RaisedButton(
  color: Colors.red,
  child: Text('Go to ScreenB'),
  onPressed: (){
    Navigator.pushNamed(context, '/b');
  })

```

1. String interpolation : 문자열 출력하거나 선언할 때 중간중간 다른 변수 끼어넣는 것

```

void main() {
  String name = 'Sean';
  print(name);

  print("Hi, $name, what's up?");
}

```

2. Collection / Generic

- Collection: 데이터들을 모아서 가지고 있는 자료구조
- Generic: Collection이 가지고 있는 데이터들의 데이터 타입을 지정
- fixed -length list: list 내의 데이터 개수가 지정된 만큼 올 수 있는
- growable list: 제한 x

```

void main(){
  var number = new List(); // ()안에 숫자를 채우면 그만큼 개수가 지정됨
}

// list 안에 다양한 타입의 데이터들이 들어올 수 있다는 의미
void main(){
  var number = new List();

  number.add(2);
  number.add('test'); // 리스트에 추가. 이외에도 함수도 추가할 수 있음.
  print(number);
}

```

```

void printNumber(List<int> a) {
    print(a);
}

void main() {
    List<dynamic> number = List();
    number.add(5);
    printNumber(number); // list 타입을 가져야함... 타입 일치 되지 않음.
}
///  

// 오류 해결. 항상 변수와 변수 타입이 일치하는지 봐야함.
void printNumber(List<int> a) {
    print(a);
}

void main() {
    List<int> number = List();
    number.add(5);
    printNumber(number);
}

```

```

void main() {
    List<String> names = List();
    names.addAll(['HO', 'JO', 'ME']);
    print(names);
}

```

generics 보충

Generics



```
class Circle {}  
class Square {}  
class SquareSlot {  
    insert (Square squareSlot) {  
    }  
}  
class CircleSlot {  
    insert (Circle circleSlot) {  
    }  
}
```

Generics



```
void main() {  
    var circleSlot = new CircleSlot();  
    circleSlot.insert(new Circle());  
    var squareSlot = new SquareSlot();  
    squareSlot.insert(new Square());  
}
```

→ 만약 다른 타입이 들어온다면 일일이 수정해줘야하나?

⇒ generics

타입을 하나로 통합, 추후에 지정할 수 있도록 해줌. → 즉 원하는 것을 골라서 지정할 수 있음.

```
void main() {  
    var circleSlot = new Slot<Circle>();  
    circleSlot.insert(new Circle());  
  
    var squareSlot = new Slot<Square>();  
    squareSlot.insert(new Square());  
    |  
}
```

```
class Circle {}   
class Square {} 
```

```
class Slot<T>{  
    insert (T shape){  
    }  
}
```