

## 2023-1\_Android

메소드 오버로딩은 가져가다 쓰는 것이고, 메소드 오버라이딩은 가져다가 !덮어쓰는것!

### 메소드 오버로딩(Method Overloading)

- 매개변수의 개수와 타입은 다르지만 이름이 같은 메소드를 여러개 개 정의하는 것
- 메소드의 기능은 같지만 매개변수의 개수와 타입이 다를 때 효율적으로 사용이 가능
- 소드의 이름을 동일한 이름으로 부여하여 메소드를 정의할 수 있도록
- 문법적으로 허용하게 되는데
- 이를 메소드 오버로딩(Method Overloading)이라고 한다.

#### 메소드 오버로딩의 예

- 가장 대표적인 것은 println 이다.
- 매개변수로 지정하는 값의 타입에 따라서 호출되는 println 메서드가 달라진다.

#### 메소드 오버라이딩의 장점

- 하나의 이름으로만 기억하면 되므로 기억하기도 쉽고 이름도 짧게 할 수 있어서 오류의 가능성을 많이 줄일 수 있다.
- 메서드의 이름만 보고 이름이 같으니, 같은 기능을 하겠구나라고 쉽게 예측할 수 있게 된다.
- 메서드의 이름을 절약할 수 있다.

#### 오버로딩을 사용하지 않았을 때

```
int sum(int a, int b) {  
    return a+b;  
}  
  
int sum_2(int a, int b, int c) {  
    return a+b+c;  
}
```

#### 오버로딩을 사용했을 때

```
int sum(int a, int b) {  
    return a+b;  
}  
  
int sum(int a, int b, int c) {  
    return a+b+c;  
}
```

### 메소드 오버라이딩(Method Overriding)

## 메소드 오버라이딩 정의

- 상위 클래스(부모 클래스)를 상속받은 하위 클래스(자식 클래스)에서 상위 클래스에 정의된 메소드를 다시 정의하는 것으로(재정의) 객체 지향 프로그래밍의 특징인 다형성을 나타낸다.
- 재정의(Overriding)는 반드시 상속 관계에 있어야 하며, 메소드의 이름, 리턴타입, 매개변수의 갯수나 타입이 완전히 일치해야 한다.
- 『static』, 『final』, 『private』 메소드는 오버라이딩(Overriding)할 수 없다.

## 메소드 오버라이딩의 장점

- 비슷한 형태의 메소드를 다시 작성하지 않아 효율성 향상
- 상속으로 인해 코드의 재사용성 향상
- 비슷하지만 다른 형태의 재정의가 가능해 다형성 확립

```
// ex) Bus 클래스는 부모클래스인 Car 클래스를 상속 받고 있다.
// Car클래스 (부모 클래스)
public class Car{
    public void run(){
        System.out.println("Car의 run메소드");
    }
}

// Bus 클래스 (자식 클래스)
public class Bus extends Car{

}

public class BusExam{
    public static void main(String args[]){
        Bus bus = new Bus();
        bus.run(); //Car의 run메소드가 실행된다.
    }
}
```

```
// Bus클래스는 부모클래스인 Car 클래스의 모양이 같은 메소드를 선언
public class Bus extends Car{
    public void run(){
        System.out.println("Bus의 run메소드");
    }
}

public class BusExam{
    public static void main(String args[]){
```

```
        Bus bus = new Bus();
        bus.run(); //Bus run메소드가 실행된다.
    }
}
```

```
/* BusExam을 실행하면 Bus클래스의 run메소드가 출력된다.
   메소드를 오버라이드 하면, 항상 자식클래스에서 정의된 메소드가 호출된다.
   오버라이딩 한다고 해서 부모의 메소드가 사라지는 것은 아니다.
   super 키워드를 이용하면, 부모의 메소드를 호출 할 수 있다.
*/
public class Bus extends Car{
    public void run(){
        super.run(); // 부모의 run()메소드를 호출
    }
}
```

---

## 4장 버튼과 에디트텍스트 (+ 소스코드)

버튼을 클릭했을 때 동작하는 Java 코드 3단계

### 3. 버튼과 에디트 텍스트

---

#### ■ 버튼을 클릭했을 때 동작하는 Java 코드 3단계

① 버튼 변수 선언

- Button mybutton;

② 변수에 버튼 위젯 대입

- mybutton = (Button) findViewById(R.id.button1);

③ 버튼을 클릭할 때 동작하는 클래스 정의

- mybutton.setOnClickListener( new View.OnClickListener() {  
    public void onClick(View v) {  
        // 동작 내용을 이 부분에 코딩  
    }  
});

---

5장 레이아웃의 종류 + 특징 + 성격 + 속성

## 2. 레이아웃의 종류

### ■ 리니어레이아웃 (선형 레이아웃)

- 왼쪽 위부터 아래쪽 또는 오른쪽으로 차례로 배치

### ■ 렐러티브레이아웃 (상대 레이아웃)

- 위젯 자신이 속한 레이아웃의 상하좌우의 위치를 지정하여 배치
- 다른 위젯으로부터 상대적인 위치 지정

### ■ 테이블레이아웃

- 위젯을 행과 열의 개수를 지정한 테이블 형태로 배열

## 2. 레이아웃의 종류

### ■ 그리드레이아웃

- 테이블레이아웃과 비슷하지만, 행 또는 열을 확장하여 다양하게 배치할 때 더 편리함

### ■ 프레임레이아웃

- 위젯들을 왼쪽 위에 일률적으로 겹쳐서 배치하여 중복해서 보이는 효과를 냄



그림 5-1 레이아웃의 종류

# 1. 레이아웃 기본 개념

## ■ 레이아웃에서 자주 사용되는 속성

- **orientation** : 레이아웃 안에 배치할 위젯의 수직 또는 수평 방향을 설정
- **gravity** : 레이아웃 안에 배치할 위젯의 정렬 방향을 좌측, 우측, 중앙으로 설정
- **padding** : 레이아웃 안에 배치할 위젯의 여백을 설정
- **layout weight** : 레이아웃이 전체 화면에서 차지하는 공간의 가중값을 설정, 여러 개의 레이아웃이 중복될 때 주로 사용
- **baselineAligned** : 레이아웃 안에 배치할 위젯을 보기 좋게 정렬

# 1. 레이아웃 기본 개념

## ■ 레이아웃

```
java.lang.Object
└ android.view.View
  └ android.widget.ViewGroup
    └ android.widget.LinearLayout
      └ android.widget.TableLayout
    └ android.widget.RelativeLayout
    └ android.widget.FrameLayout
    └ android.widget.GridLayout
```

레이아웃 계층도

- ViewGroup 클래스로부터 상속받으며 내부에 무엇을 담는 용도로 사용
- 레이아웃 중에서 가장 많이 사용되는 것은 리니어레이아웃(LinearLayout)

```
public class MainActivity extends AppCompatActivity {

    LinearLayout baseLayout;
    Button button1;

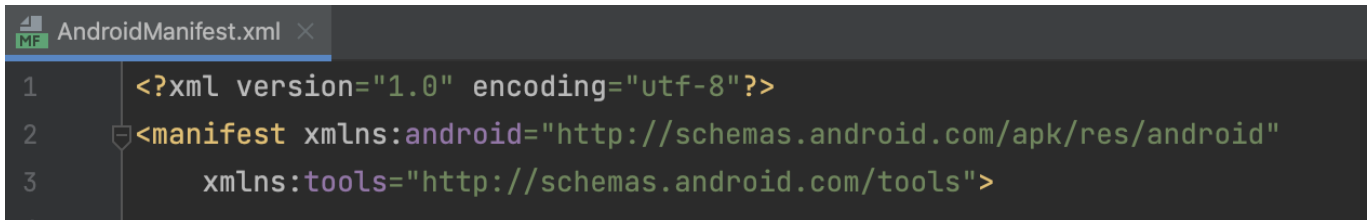
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        setTitle("배경색 바꾸기");

        baseLayout = (LinearLayout) findViewById(R.id.baseLayout);
        button1 = (Button) findViewById(R.id.button1);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        super.onCreateOptionsMenu(menu);
        MenuInflater mInflater = getMenuInflater();
        mInflater.inflate(R.menu.menu1, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.itemRed:
                baseLayout.setBackgroundColor(Color.RED);
                return true;
            case R.id.itemGreen:
                baseLayout.setBackgroundColor(Color.GREEN);
                return true;
            case R.id.itemBlue:
                baseLayout.setBackgroundColor(Color.BLUE);
                return true;
            case R.id.subRotate:
                button1.setRotation(45);
                return true;
            case R.id.subSize:
                button1.setScaleX(2);
                return true;
        }
        return false;
    }
}
```



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:tools="http://schemas.android.com/tools">
```

매니페스트 파일에는 많은 정보를 담을 수 있지만 그중에서도 특히 선언되어야 하는 정보가 있습니다.

- 앱의 패키지 이름
- 앱에서 사용되는 컴포넌트(액티비티, 서비스, 브로드캐스트 리시버, 콘텐츠 프로바이더)
- 권한(Permission)
- 앱에서 요구하는 하드웨어와 소프트웨어 특징

---

## 인의예지신 중도

---



## 인(仁)은

측은지심(惻隱之心)으로 불쌍한 것을 보면 가엾게 여겨 정을 나누고  
자 하는 마음

## 의(義)는

수오지심(羞惡之心)으로 불의를 부끄러워하고 악한 것은 미워하는  
마음

## 예(禮)는

사양지심(事讓之心)으로 자신을 낮추고 겸손해하며 남을 위해 사양  
하고 배려할 줄 아는 마음

## 지(智)는

시비지심(是非之心)으로 옳고 그름을 가릴 줄 아는 마음

## 신(信)은

광명지심(光明之心)으로 중심을 잡고 항상 가운데에 바르게 위치해  
밝은 빛을 냄으로써 믿음을 주는 마음