상속

기반클래스

다른 클래스로부터 상속을 받지 않은 클래스

```swift
class Person {
    var name: String = ""

    func selfIntroduce() {
        print("저는 \(name)입니다")
    }

    // final 키워드를 사용하여 재정의를 방지할 수 있습니다
    final func sayHello() {
        print("hello")
    }

    // 타입 메서드
    // 재정의 불가 타입 메서드 - static
    static func typeMethod() {
        print("type method - static")
    }

    // 재정의 가능 타입 메서드 - class
    class func classMethod() {
        print("type method - class")
    }

    // 재정의 가능한 class 메서드라도
    // final 키워드를 사용하면 재정의 할 수 없습니다
    // 메서드 앞의 `static`과 `final class`는 똑같은 역할을 합니다
    final class func finalCalssMethod() {
        print("type method - final class")
    }
}
```

재정의시 override 키워드 사용
부모클래스의 특성을 자식 클래스에서 재정의했을때 다시 부모클래스의 특성을 활용하고 싶을 때 super프로퍼티를 사용하면된다.

```swift
// Person을 상속받는 Student
class Student: Person {
    var major: String = ""

    override func selfIntroduce() {
        print("저는 \(name)이고, 전공은 \(major)입니다")
    }

    override class func classMethod() {
        print("overriden type method - class")
    }

}
```

클래스의 이니셜라이저

지정 이니셜라이저

- 필요에 따라 부모클래스의 이니셜라이저 호출
- 이니셜라이저가 정의된 클래스의 모든 프로퍼티를 초기화해야됨
- 모든 클래스는 하나 이상의 지정 이니셜라이저를 갖음
- 부모클래스의 지정 이니셜라이저가 자식클래스의 지정 이니셜라이저 역할을 충분히 한다면 자식 클래스는 지정 이니셜라이저를 갖지않아도 된다. -> 부모 클래스로부터 물려받은 저장 프로퍼티를 제외하고 옵셔널 저장프로퍼티 외에 다른 프로퍼티가 없을 때

편의 이니셜라이저

- 지정 이니셜라이저를 자신의 내부에서 호출
- 지정 이니셜라이저의 매개변수가 많아 외부에서 일일이 전달하기 어렵거나 특정 목적에 사용하기 위해 편의 이니셜라이저 선언
- 더 적은 입력값으로 초기화를 편리하게 할 수 있다.

```swift
class Student : Person {
    var major : String

    override init(name: String, age: Int) {
        self.major = "Swift"
        super.init(name: name, age: age)
    }

    convenience init(name: String) {
        self.init(name: name, age: 7)
    }
}
```

클래스의 초기화 위임

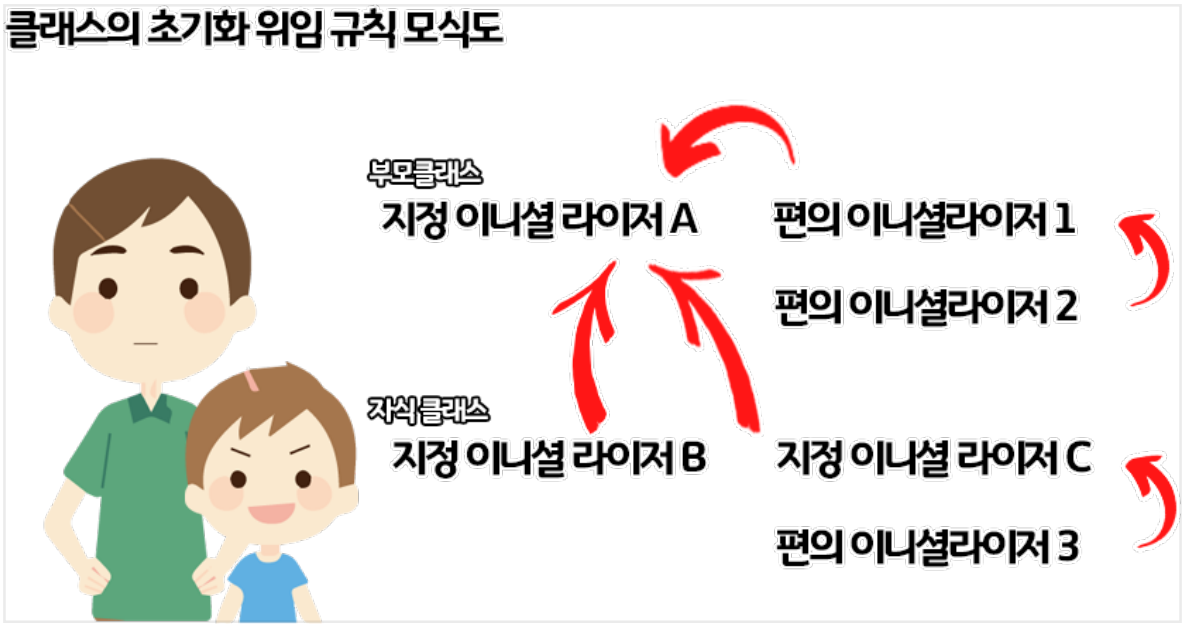
지정 이니셜라이저와 편의 이니셜라이저 사이의 관계에 대한 규칙

1. 자식클래스의 지정 이니셜라이저는 부모클래스의 지정 이니셜라이저를 반드시 호출해야한다.
2. 편의 이니셜라이저는 자신을 정의한 클래스의 다른 이니셜라이저를 반드시 호출해야한다.
3. 편의 이니셜라이저는 궁극적으로 지정 이니셜라이저를 반드시 호출해야 한다.

다시 정리하면

- 누군가는 지정 이니셜라이저에게 초기화를 반드시 위임합니다.
- 편의 이니셜라이저는 초기화를 반드시 누군가에 위임합니다.

이때 누군가는 다른 지정 이니셜라이저 또는 편의 이니셜라이저를 뜻함

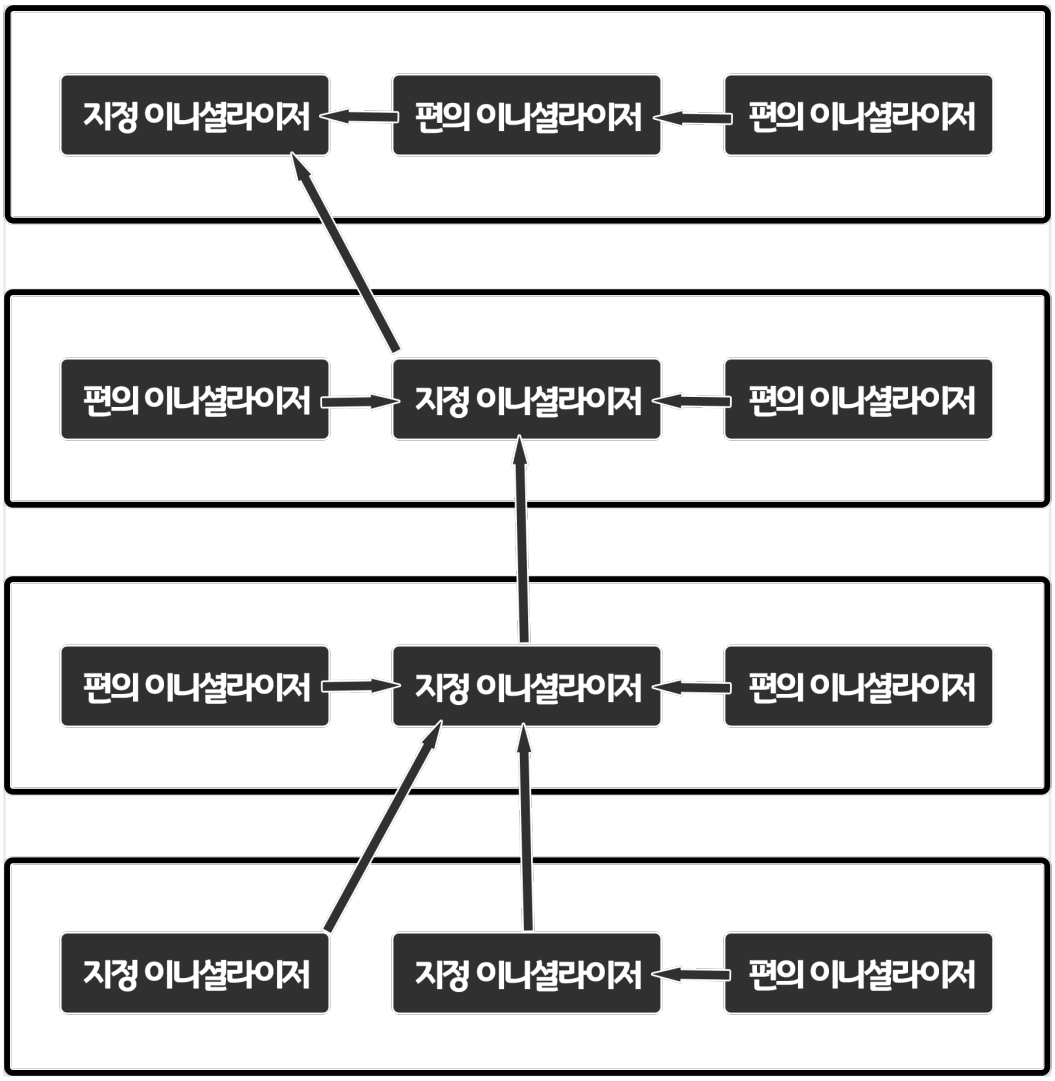


부모클래스는 하나의 지정 이니셜라이저(A)와 두 개의 편의 이니셜라이저(1, 2)가 존재한다.

부모클래스의 편의 이니셜라이저(2)는 다른 편의 이니셜라이저(1)을 호출하고
그 편의 이니셜라이저(1)는 궁극적으로 지정 이니셜라이저(A)를 호출한다. - 규칙 2,3 만족

자식클래스에는 두 개의 지정 이니셜라이저(B, C)와 편의 이니셜라이저(3)가 존재한다. 편의 이니셜라이저(3)은 지정 이니셜라이저(C)를 호출하고, 편의 이니셜라이저는 자신의 클래스에 구현된 아니셜라이저만 호출 가능하므로 부모클래스의 이니셜라이저는 호출 불가하다 - 규칙 2, 3 만족

두 지정 이니셜라이저 모두 부모 클래스의 지정 이니셜라이저(A)를 호출한다. - 규칙 1 만족



조금 더 복잡한 모식도로 지정 이니셜라이저가 어떻게 클래스의 이니셜라이저 중 기둥 역할을 하는지 조금 더 쉽게 알 수 있다.

2단계 초기화

클래스의 초기화는 2단계를 거칩니다.

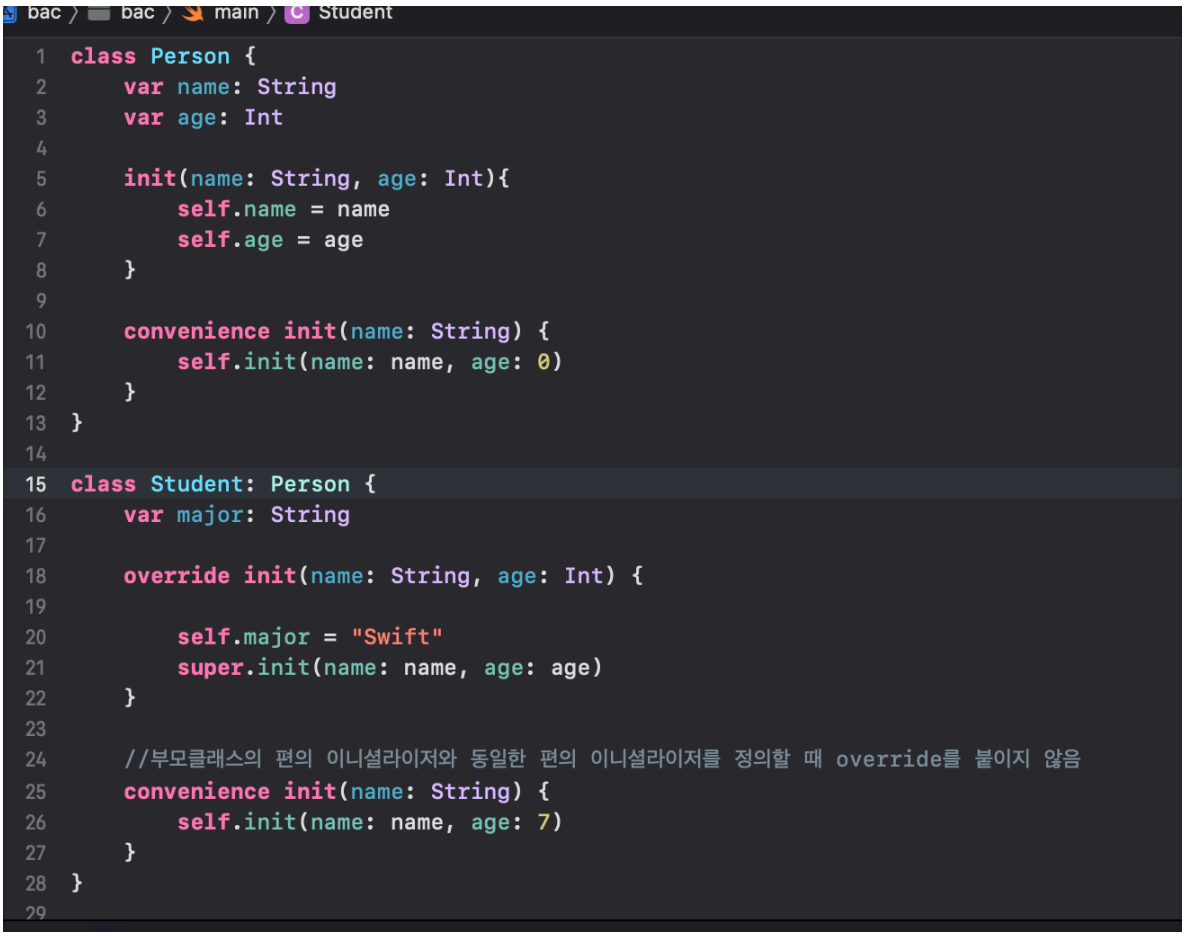
1단계는 클래스에 정의한 각각의 저장 프로퍼티에 초기값이 할당
2단계는 자식 및 부모클래스의 저장 프로퍼티 값을 변경한다. 그리고 메서드 및 연산 프로퍼티를 사용하여 초기화를 진행한다

2단계 초기화시 네 가지 안전확인

1. 자식클래스의 지정 이니셜라이저가 부모클래스의 이니셜라이저를 호출하기 전에 자신의 프로퍼티를 모두 초기화했는지 확인합니다.
2. 자식클래스의 지정 이니셜라이저는 상속받은 프로퍼티에 값을 할당하기 전에 반드시 부모클래스의 이니셜라이저를 호출해야 합니다.
3. 펀의 이니셜라이저는 자신의 클래스에 정의한 프로퍼티를 포함하여 그 어떤 프로퍼티라도 값을 할당하기 전에 다른 이니셜라이저를 호출해야 합니다.
4. 초기화 1단계를 마치기 전까지는 이니셜라이저는 인스턴스 메서드를 호출할 수 없습니다. 또 인스턴스 프로퍼티의 값을 읽어 들일 수도 없습니다. self 프로퍼티를 자신의 인스턴스를 나타내는 값으로 활용할 수도 없습니다.

```swift
class Person {
    var name: String
    var age: Int

    init(name: String, age: Int){
        self.name = name
        self.age = age
    }
}

class Student: Person {
    var major: String

    init(name: String, age: Int, major: String) {
        //자식 클래스의 속성 값에 값 할당 안전확인 1번 조건 충족
        self.major = "Swift"

        //자식클래스의 지정 이니셜라이저는
        //부모클래스의 지정 이니셜라이저를 반드시 호출 해야함
        //안전 확인 2번 충족
        super.init(name: name, age: age)
    }

    convenience init(name: String) {
        //편의 이니셜라이저는 따로 차후에 값을 할당할 프로퍼티가 없고, 다른 이니셜라이저를 호출했으므로 3번 조건 충족
        //어디에서도 인스턴스 메서드를 호출하거나 인스턴스 프로퍼티의 값을 읽어오지 않았으므로 4번 충족
        self.init(name: name, age: 7, major: "")
    }
```

이니셜라이저 상속 및 재정의



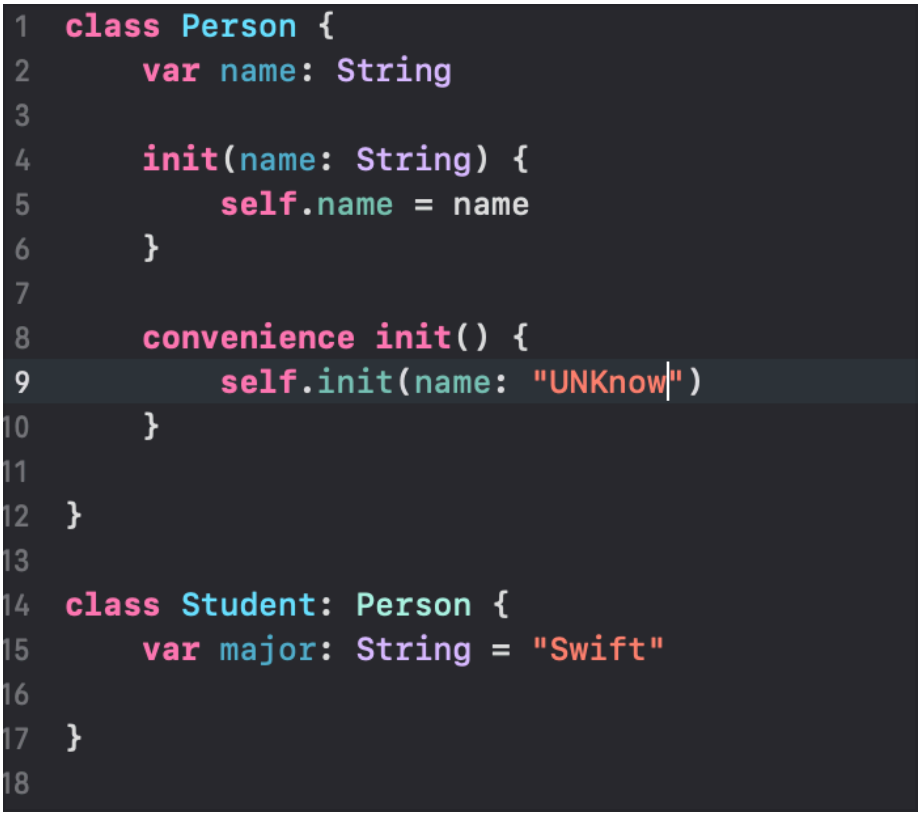
부모클래스의 편의 이니셜라이저와 동일한 편의 이니셜라이저를 정의할 때 override를 붙이지 않음
반대로 지정 이니셜라이저는 재정의를 위해 override 수식어를 사용한 것을 볼 수 있다.
기본 이니셜라이저 외에 지정 이니셜라이저를 자식클래스에서 동일한 이름으로 정의 하려면 재정의를 위한 override를 명시해줘야됨

이니셜라이저 자동 상속

부모클래스의 이니셜라이저를 자동 상속 받기 위해서는 두 가지 규칙이 있다.

자식클래스에서 프로퍼티 기본값을 모두 제공한다고 가정

1. 자식 클래스에서 별도의 지정 이니셜라이저를 구현하지 않는다면, 부모클래스의 지정 이니셜라이저가 자동으로 상속된다.
2. 부모클래스의 지정 이니셜라이저를 모두 재정의하여 부모클래스와 동일한 지정 이니셜라이저를 모두 사용 할 수 있는 상황



```swift
class Person {
    var name: String

    init(name: String) {
        self.name = name
    }

    convenience init() {
        self.init(name: "UNKnow")
    }

}

class Student: Person {
    var major: String = "Swift"

}
```

규칙 1 부합

요구 이니셜라이저

- 모든 자식 클래스에서 반드시 구현해야 하는 이니셜라이저
- 클래스의 이니셜라이저 앞에 required 수식어를 사용한다. 자식클래스에서 구현할 때도 required 수식어 필수
- required는 기본적으로 override를 포함한다. (자식클래스에서 요구 이니셜라이저를 재정이 할 때는 override 대신 required 수식어 사용)