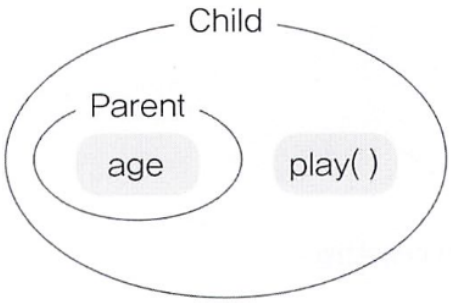
# 07장(객체지향언어 2)

## 01. 상속

### 상속

- 기존 클래스를 재사용하여 새로운 클래스를 작성하는 것...(차암.. 어렵게도 설명해놨네...)
- **extends 예약어**를 사용
- 생성자와 초기화 블럭은 상속되지 않음, 멤버(변수, 함수)만 상속
- **조상 클래스** → 부모클래스, 상위클래스, 기반클래스
- **자손 클래스** → 자식클래스, 하위클래스, 파생된클래스
- 자손 클래스의 멤버 개수 ≥ 조상클래스의 멤버 개수

### 상속의 형태

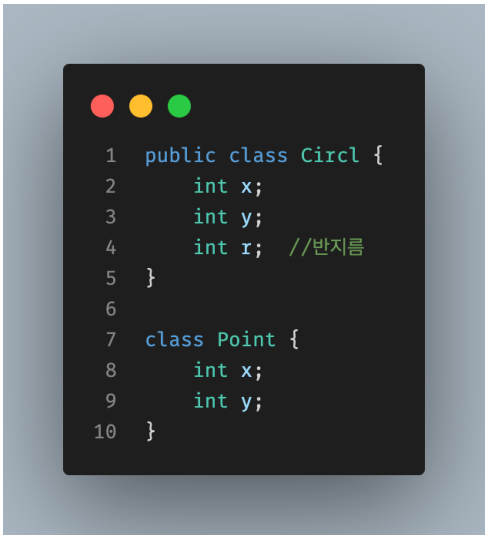



| 클래스 | 클래스의 멤버 |
|---|---|
| Parent | age |
| Child | age, play( ) |

### 포함관계

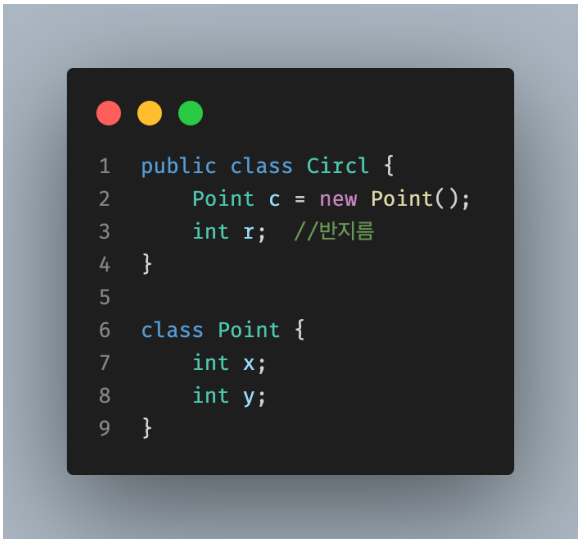
- 상속 이외에도 클래스를 재사용하는 방법

ex) 좌표와 반지름



```java
public class Circl {
    int x;
    int y;
    int r;  //반지름
}

class Point {
    int x;
    int y;
}
```



```java
public class Circl {
    Point c = new Point();
    int r;  //반지름
}

class Point {
    int x;
    int y;
}
```

### 클래스간의 관계 결정

- 클래스를 가지고 문장을 만들었을 때
  - **'~은 ~이다.' 라는 문장이 성립하면 → 상속관계**
  - **'~은 ~을 가지고 있다.' 라는 문장이 성립하면 → 포함관계**

### 단일상속(single inheritance)

- C++에서는 다중상속이 가능하지만, 자바에서는 오직 단일 상속만 허용됨

### Object 클래스

- 모든 클래스의 조상
- 모든 클래스의 최상위 클래스

---

# 02. 오버라이딩

### !!주의!! 오버로딩 ≠ 오버라이딩

### 오버라이딩

- 조상 클래스로 부터 상속받은 메서드의 내용을 재정의 하는 것
- 조건
  - **이름이 같아야함**
  - **매개변수가 같아야함**

- **반환타입이 같아야함**

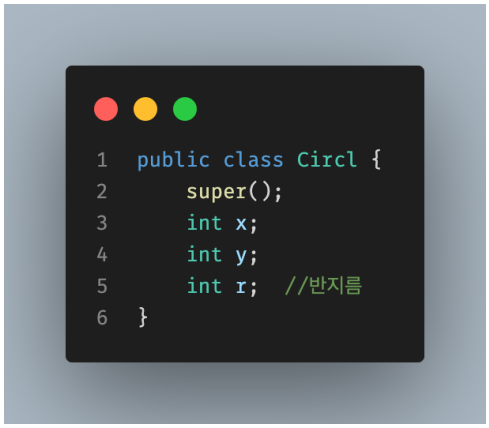
  → 1.5 이후 반환타입을 자손 클래스의 타입으로 변경하는 것은 가능해짐
- **접근 제어자는 조상 클래스의 메서드보다 좁은 범위 불가능**
- **예외 선언은 조상 클래스보다 많은 수로는 불가능**
- **인스턴스메서드를 static메서드로 or 그 반대의 경우로도 변경할 수 없음**

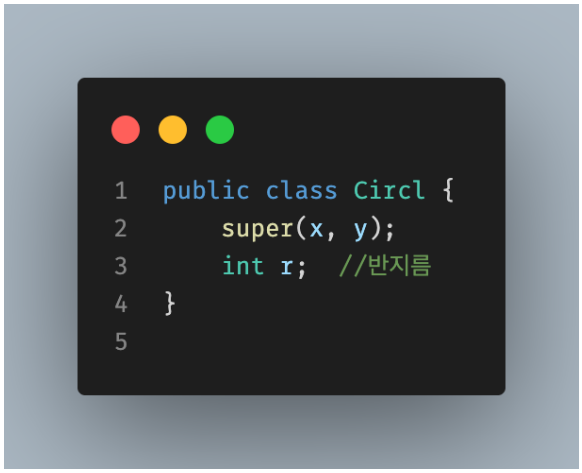
### super

- 자손 클래스에서 조상클래스로부터 상속받은 것을 참조하는데 사용되는 참조 변수
- super대신 this를 사용 할 수 있음
- **서로 구별해야 하는 경우에만 super를 사용하는 것이 좋음**
- static 사용불가 인스턴스에서만 사용가능

### super( )

- **this( )와 같은 기능으로 조상 클래스의 생성자 호출시 사용**
- 조상 클래스에 없는 생성자 패턴을 호출하면 에러가 발생함




---

# 03. Package와 import

### 패키지

- Class + Interface로 구성되어 있는 디렉토리

### import문

- 사용하고자 하는 패키지를 미리 명시하여 사용되는 클래스이름에서 패키지명을 생략
- **import 패키지명.클래스명;**
- **import 패키지명.*;**
  → 해당 피키지에 있는 모든 클래스를 임폴트 시킨다는 뜻
  → 실행 시 성능상의 차이는 전혀 없다(....? 정말?)

### static import문

- **static** 멤버를 호출할 때 클래스 이름을 생략하게 할 수 있다.

```
System.out.println(Math.random());   ←——→   out.println(random());
```

---

# 04. 제어자

| | |
|---|---|
| 접근 제어자 | public, protected, default, private |
| 그 외 | static, final, abstract, native, transient, synchronized, volatile, strictfp |

### static

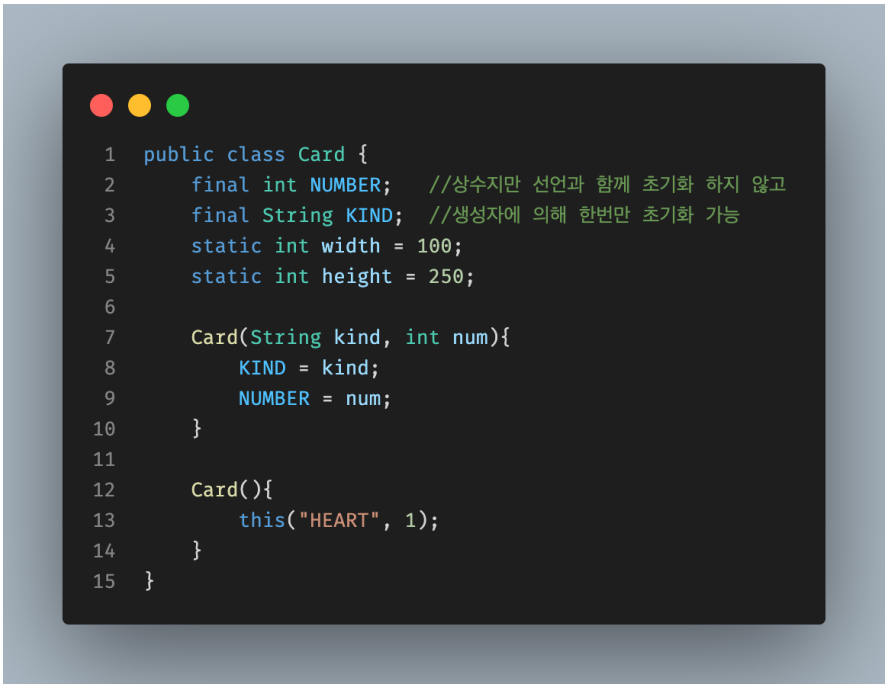
| 대상 | 의미 |
|---|---|
| 멤버변수 | - 모든 인스턴스에 공통적으로 사용되는 클래스 변수가 됨 - 인스턴스 생성하지 않고 사용 가능 - 클래스가 메모리에 로드 될 때 생성 |
| 메서드 | - 인스턴스 생성하지 않고 호출이 가능 - 구현부({ })에서 인스턴스멤버들을 직접 사용 불가능 |

### final

| 대상 | 의미 |
|---|---|
| 클래스 | - 변경될 수 없는 클래스, 확장할 수 없는 클래스가 됨 - 다른 클래스의 조상이 될 수 없음 |
| 메서드 | - 변경될 수 없는 메서드, 오버라이딩을 통해 재정의 불가능 |
| 멤버변수 | - 변경할 수 없는 상수가 됨 |
| 지역변수 | - 변경할 수 없는 상수가 됨 |

### 생성자를 이용한 final 멤버 변수의 초기화

- **인스턴스 변수의 경우 생성자를 이용하여 초기화 되도록 할 수 있음**



```java
public class Card {
    final int NUMBER;   //상수지만 선언과 함께 초기화 하지 않고
    final String KIND;  //생성자에 의해 한번만 초기화 가능
    static int width = 100;
    static int height = 250;

    Card(String kind, int num){
        KIND = kind;
        NUMBER = num;
    }

    Card(){
        this("HEART", 1);
    }
}
```

## abstract

| 대상 | 의미 |
|---|---|
| 클래스 | - 클래스 내에 추상 메서드가 선언되어 있음을 의미 |
| 메서드 | - 선언부만 작성하고 구현부는 자성하지 않음 메서드 → 오버라이딩 해야함 |

## 접근 제어자

| 제어자 | 같은 클래스 | 같은 패키지 | 자손 클래스 | 전체 | 사용가능한 대상 |
|---|---|---|---|---|---|
| public | O | O | O | O | 메서드, 멤버변수, 클래스 |
| protected | O | O | O | X | 메서드, 멤버변수 |
| (default) | O | O | X | X | 메서드, 멤버변수, 클래스 |
| private | O | X | X | X | 멤버변수 |

## 접근 제어자를 이용한 캡슐화

- 데이터가 유효한 값을 유지하도록 함
- 비밀번호와 같은 데이터를 외부로 노출 및 변경하지 못하도록 막음

## 생성자의 접근 제어자

- 외부에서 다른 클래스가 접근할 수 없어 인스턴스를 생성할 수 없게됨

- 클래스 내부에서는 인스턴스 생성 가능
- 따라서 public메서드를 이용하여 이 클래스의 인스턴스를 사용하도록 할 수 있음
- 이 메서드는 public과 동시에 static이어야 함 → ex) **회사와 부서가 있을때 회사는 싱글톤...**

```
1  public class Singleton {
2          ...
3      private static Singleton s = new Singleton();
4      private Singeton(){
5          ...
6      }
7
8      //인스턴스를 생성하지 않고도 호출할 수 있어야 하므로 static 이어야함
9      public static Singleton getInstance(){
10         return s;
11     }
12         ...
13 }
```

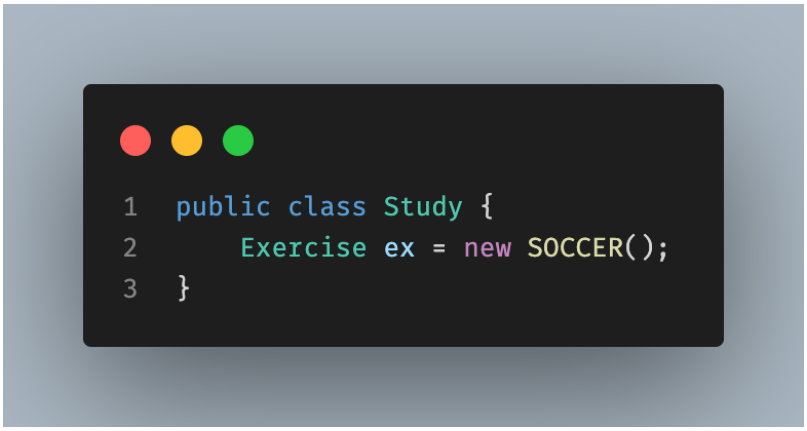
---

# 05. 다형성

### 다형성

- 조상 클래스 타입의 참조변수로 자손클래스의 인스턴스를 참고 할수 있도록 함

  → 상속받아 하나의 참조타입으로 처리가 가능함
- **참조변수 사용가능 멤버 개수 ≤ 인스턴스의 멤버 개수**



```
1  public class Study {
2      Exercise ex = new SOCCER();
3  }
```

### Up-Casting

- **자손타입 → 조상타입 : 형변환 생략 가능**
- 조상이 갖고있는 멤버가 자손보다 적을것을 알기 때문에 생략 가능한 것

### Down-Cating

- **조상타입 → 자손타입 : 형변환 생략 불가**
- **instanceof**를 이용하여 변환하면 더욱 안전함

### 형변환 체크사항

- 형변환은 참조변수의 타입을 변환하는 것이지 인스턴스가 변환하는 것은 아님
  → 인스턴스에게 아무런 영향을 미치지 않음
- **단지, 참조하고 있는 인스턴스에서 사용할 수 있는 멤버의 범위를 조절하는 것 뿐**

### 참조변수와 인스턴스의 연결 (이름 참 이상하게 지음...)

- 부모는 자식꺼를 사용하지 못하기기 때문에...!!!!!
  **부모타입의 참조변수** → 부모의 메서드 사용
  **자식타입의 참조변수** → 부모의 메서드를 오버라이딩한 메서드를 사용

---

# 06. 추상클래스

### 추상클래스 (abstract)

- **미완성 설계도라고 할 수 있음**
- **abstract 예약어**로 보고 추상메서드가 있으니 상속을 통해 구현해주어야 한다는 것을 명시

---

# 07. 인터페이스

### 인터페이스

- 일종의 추상클래스
- 다중상속이 가능 → 가능은하지만 구현하는 경우는 거의 없음...
- 인터페이스 끼리는 **extends 예약어**를 사용해서 상속받는다.
- 클래스에서 구현할 때는 **implements 예약어**를 사용
- **static final만 정의할 수 있으므로 조상클래스의 멤버변수와 충돌하는 경우는 거의 없다.**
  → **충돌이 발생하더라도 클래스 이름을 붙여서 구분이 가능함**

### 인터페이스를 이용한 다형성

- 추상 클래스 or 인터페이스들을 메서드를 이용하여 반환타입에 생성자를 넣어 생성

  → ex) Calendar Class

### 인터페이스의 장점

- 개발시간을 단축시킬 수 있다.
- 일관되고 정형화된 프로그램의 개발이 가능 → 메서드의 이름이 통일
- 인터페이스를 공통적으로 구현(상속)하도록 하여 관계를 맺어 줄 수 있음
- 한 클래스의 변경이 관련된 다른 클래스에 영향을 미치지 않는 독립적인 프로그래밍이 가능

### 디폴트 메서드

- JDK1.8부터 인터페이스에 추가 할 수 있게됨
- **인터페이스에 새로운 메서드 추가시!!**

  → 상속한 클래스들을 전부 수정할 필요가 없어짐

  → 앞에 **키워드 default**를 붙이며 **블록({ })**이 있어야함 : **접근제어자는 public**이며 생략 가능
- 추가된 디폴트 메서드가 기존의 메서드와 이름이 중복되어 충돌 발생시 해결방법

  → 그냥 필요한 쪽의 메서드와 같은 내용으로 오버라이딩 해버려도됨

> 1. 여러 인터페이스의 디폴트 메서드 간의 충돌
>  – 인터페이스를 구현한 클래스에서 디폴트 메서드를 오버라이딩해야 한다.
> 2. 디폴트 메서드와 조상 클래스의 메서드 간의 충돌
>  – 조상 클래스의 메서드가 상속되고, 디폴트 메서드는 무시된다.

### static 메서드

- JDK1.8부터 인터페이스에 추가 할 수 있게됨
- 인스턴스와 관계가가 없는 독립적인 메서드이기 때문에 넣지 못할 이유가 없었다...

---

# 08. 내부클래스

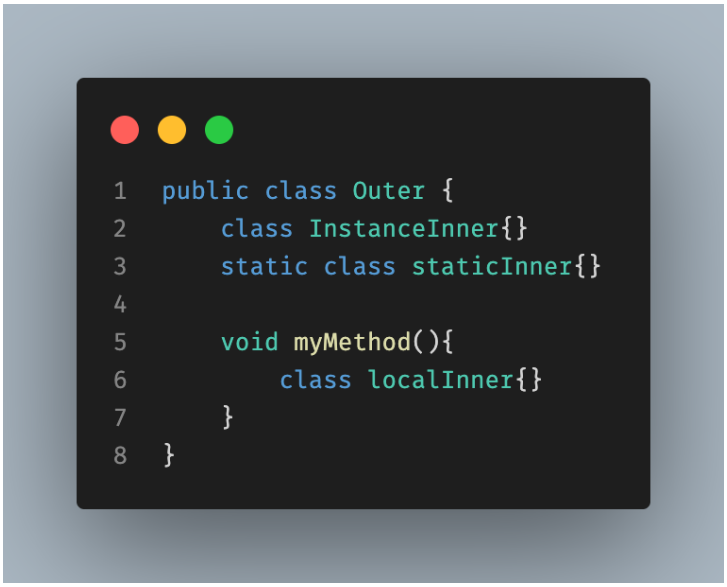
### 내부클래스

- 클래스안에 클래스를 생성하는 것
- 코드의 복잡성을 줄일 수 있음 → **캡슐화**

### 내부 클래스의 종류

| 내부 클래스 | 특징 |
| --- | --- |
| 인스턴스클래스 (instance class) | - 외부클래스의 멤버변수 위치에 선언하며, 외부 클래스의 인스턴스 멤버처럼 다루어짐 - 주로 외부 클래스의 인스턴스 멤버들과 관련된 작업에 사용됨 |
| 스태틱 클래스 (static class) | - 외부클래스의 멤버변수 위치에 선언하며, 외부 클래스의 static 멤버처럼 다루어짐 - 주로 외부 클래스의 static 멤버, 특히 static 메서드에서 사용될 목적으로 선언 |
| 지역 클래스 (local class) | - 외부 클래스의 메서드나 초기화블럭 안에 선언 - 선언된 영역 내부에서만 사용가능 |
| 익명 클래스 (anoymous class) | - 클래스의 선언과 객체의 생성을 동시에 하는 이름없는 클래스 - 단발성(일회용) |

### 내부 클래스의 선언



```
1  public class Outer {
2      class InstanceInner{}
3      static class staticInner{}
4
5      void myMethod(){
6          class localInner{}
7      }
8  }
```

### 내부 클래스의 제어자와 접근성

- **인스턴스 멤버와 static 멤버 간의 규칙이 내부 클래스에도 똑같이 적용**

  ex) static 클래스만 static멤버를 정의할 수 있음

- static 클래스에서는 인스턴스 클래스의 멤버들을 객체생성 없이 사용할수 없다.

```java
public class InnerEx2 {
    class InstanceInner {}
    static class StaticInner {}

    // 인스턴스멤버 간에는 서로 직접 접근이 가능하다'
    InstanceInner iv = new InstanceInner() ;
    // static 멤버 간에는 서로 직접 접근이 가능하다.
    static StaticInner Cv = new StaticInner();

    static void staticMethod() {
        //static멤버는인스턴스멤버에직접접근할수없다
        InstanceInner objet = new InstanceInner();  //FIXME: 오류발생

        StaticInner object2 = new StaticInner();

        // 굳이 접근하려면 아래와 같이 객체를 생성해야 한다.
        // 인스턴스클래스는 외부 클래스를 먼저 생성해야만 생성할 수 있다.
        InnerEx2 outer = new InnerEx2 ();
        InstanceInner Objl = outer.new InstanceInner();
    }

    void instanceMethod(){
        //인스턴스메서드에서는인스턴스멤버와static멤버모두접근가능하다
        InstanceInner obj1 = new InstanceInner();
        StaticInner obj2 = new StaticInner();

        //메서드내에 지역적으로 선언된 내부클래스는외 부에서접근할수없다.
        LocalInner lv = new LocalInner();   //FIXME: 오류발생
    }

    void myMethod() {
        class LocalInner{}
        LocalInner lv = new LocalInner();
    }
}
```

**익명 클래스**

- **단발성(일회용)**
- 익명 클래스는 이름이 없기 때문에 '**외부 클래스명$숫자.class**'의 형식으로 클래스파일명이 결정됨