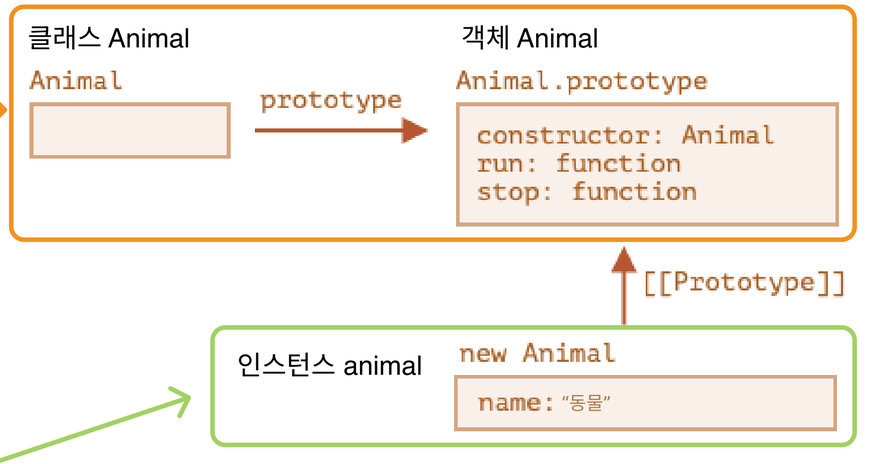
# 클래스의 상속

extends 생성자 오버라이딩 메서드 오버라이딩 super

# extends

## 부모 class Animal

```
1  class Animal {
2      constructor(name) {
3          this.speed = 0;
4          this.name = name;
5      }
6      run(speed) {
7          this.speed = speed;
8          alert(`${this.name} 은/는 속도 ${this.speed}로 달립니다.`);
9      }
10     stop() {
11         this.speed = 0;
12         alert(`${this.name} 이/가 멈췄습니다.`);
13     }
14 }
15
16 let animal = new Animal("동물");
```



# extends

## 자식 class Rabbit

```
class Rabbit extends Animal {
    hide() {
        alert(`${this.name} 이/가 숨었습니다!`);
    }
}

let rabbit = new Rabbit("흰 토끼");

rabbit.run(5); // 흰 토끼 은/는 속도 5로 달립니다.
rabbit.hide(); // 흰 토끼 이/가 숨었습니다!
```

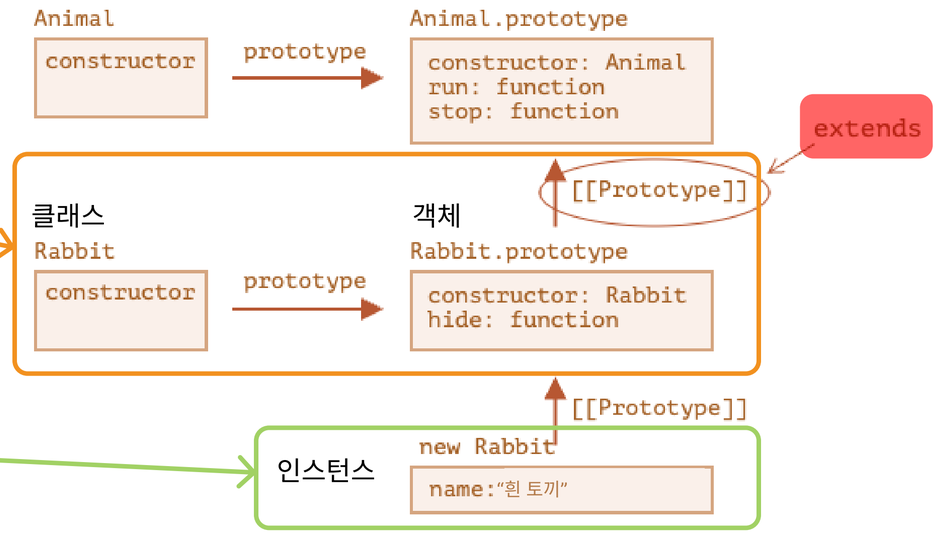
**class Child “extends” Parents**

extends 키워드를 통해
부모 class에 있는 메서드를 상속받음

# extends

## 자식 class Rabbit

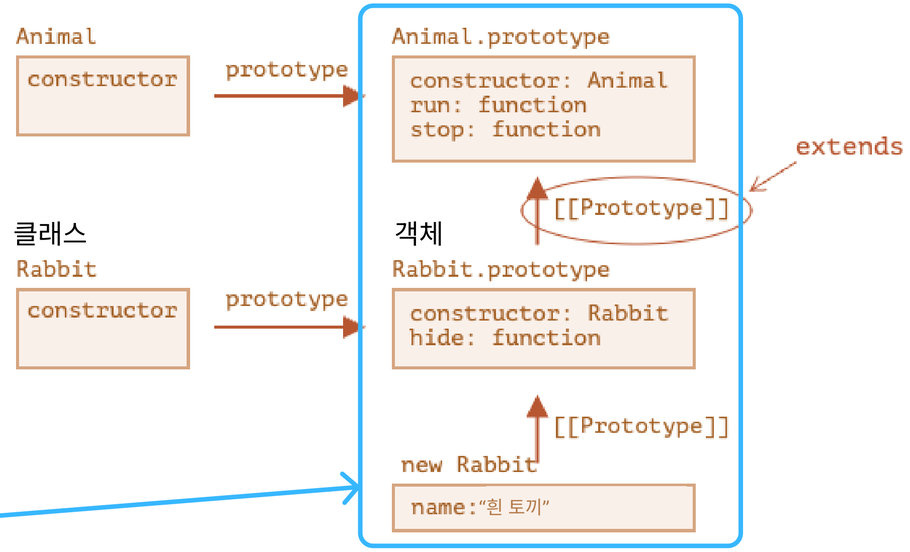
```
1  class Rabbit extends Animal {
2      hide() {
3          alert(`${this.name} 이/가 숨었습니다!`);
4      }
5  }
6
7  let rabbit = new Rabbit("흰 토끼");
8
9  rabbit.run(5); // 흰 토끼 은/는 속도 5로 달립니다.
10 rabbit.hide(); // 흰 토끼 이/가 숨었습니다!
```



# extends

## 자식 class Rabbit

```
1  class Rabbit extends Animal {
2      hide() {
3          alert(`${this.name} 이/가 숨었습니다!`);
4      }
5  }
6
7  let rabbit = new Rabbit("흰 토끼");
8
9  rabbit.run(5); // 흰 토끼 은/는 속도 5로 달립니다.
10 rabbit.hide(); // 흰 토끼 이/가 숨었습니다!
```



특별한 사항이 없으면 class Rabbit은 class Animal에 있는 메서드를 **‘그대로’ 상속**받는다.

그런데 Rabbit에서 **부모 클래스인 animal에 있던 메서드를 자체적으로 정의**하면,
상속받은 메서드가 아닌 자체 메서드가 사용된다.

## 생성자 • 메서드 오버라이딩

자식 클래스가 자신의 슈퍼or **부모 클래스**들 중 하나에 의해
**이미 제공된 생성자나 메서드**를 특정한 형태로 구현하는 것을
제공하는 언어의 특징

## 생성자 • 메서드 오버라이딩

자식 클래스가 자신의 슈퍼or **부모 클래스**들 중 하나에 의해
**이미 제공된 생성자나 메서드**를 특정한 형태로 구현하는 것을
제공하는 언어의 특징

## super

super.method(...) - 부모 클래스에 정의된 **메서드**를 호출
super(...) - 부모 **생성자**를 호출, 자식 생성자 내부에서만 사용가능

# super 생성자 오버라이딩

```
class Rabbit extends Animal {
    //생성자 오버라이딩
    constructor(name, earLength) {
        //기존메서드
        super(name);
        //새로운 생성자 추가
        this.earLength = earLength;
    }

    //메서드 오버라이딩
    //새로운 메서드 추가
    hide() {
        alert(`${this.name}가 숨었습니다!`);
    }

    //기존 메서드 수정
    stop() {
        super.stop(); // 부모 클래스의 stop을 호출해 멈추고,
        this.hide(); // 숨습니다.
    }
}
```

```
let rabbit = new Rabbit("흰 토끼", 10);

//생성자 오버라이딩
alert(rabbit.speed); // 0 (speed도 출력가능)
alert(rabbit.name); // 흰 토끼
alert(rabbit.earLength); // 10

//메서드 오버라이딩
rabbit.run(5); // 흰 토끼가 속도 5로 달립니다.
rabbit.stop(); // 흰 토끼가 멈췄습니다. 흰 토끼가 숨었습니다!
```

super(name) - 부모 생성자 name를 호출
-> rabbit 인스턴스에서 인자로 받은 값 “흰 토끼"를 name으로 할당

- class Animal에 있는 speed 접근 가능
- class Rabbit에 추가된 earLength도 접근 가능

# super 메서드 오버라이딩

```
class Rabbit extends Animal {
    //생성자 오버라이딩
    constructor(name, earLength) {
        //기존메서드 수정
        super(name);
        //새로운 생성자 추가
        this.earLength = earLength;
    }

    //메서드 오버라이딩
    //새로운 메서드 추가
    hide() {
        alert(`${this.name}가 숨었습니다!`);
    }

    //기존 메서드 수정
    stop() {
        super.stop(); // 부모 클래스의 stop을 호출해 멈추고,
        this.hide(); // 숨습니다.
    }
}
```

```
let rabbit = new Rabbit("흰 토끼", 10);

//생성자 오버라이딩
alert(rabbit.speed); // 0 (speed도 출력가능)
alert(rabbit.name); // 흰 토끼
alert(rabbit.earLength); // 10

//메서드 오버라이딩
rabbit.run(5); // 흰 토끼가 속도 5로 달립니다.
rabbit.stop(); // 흰 토끼가 멈췄습니다. 흰 토끼가 숨었습니다!
```

super.stop() - 부모 메서드 stop()를 호출
-> rabbit.stop() 부모 메서드 stop()을 호출해 멈추고,
hide()까지하는 Rabbit의 stop()으로 수정

- class Animal에 있는 run() 호출 가능

# super 콜백함수로서의 super

### 일반함수

```
1  class Rabbit extends Animal {
2      stop() {
3          //오류발생 Unexpected super
4          setTimeout(function () {super.stop();}, 1000);
5      }
6  }
```

### 화살표함수

```
1  class Rabbit extends Animal {
2      stop() {
3          // 1초 후에 부모 stop을 호출
4          setTimeout(() => super.stop(), 1000);
5      }
6  }
```

setTimeout의 콜백함수로 super를 사용할 때
일반 함수로는 사용할 수 없고, **화살표함수를 사용**해야한다.

# THANK YOU