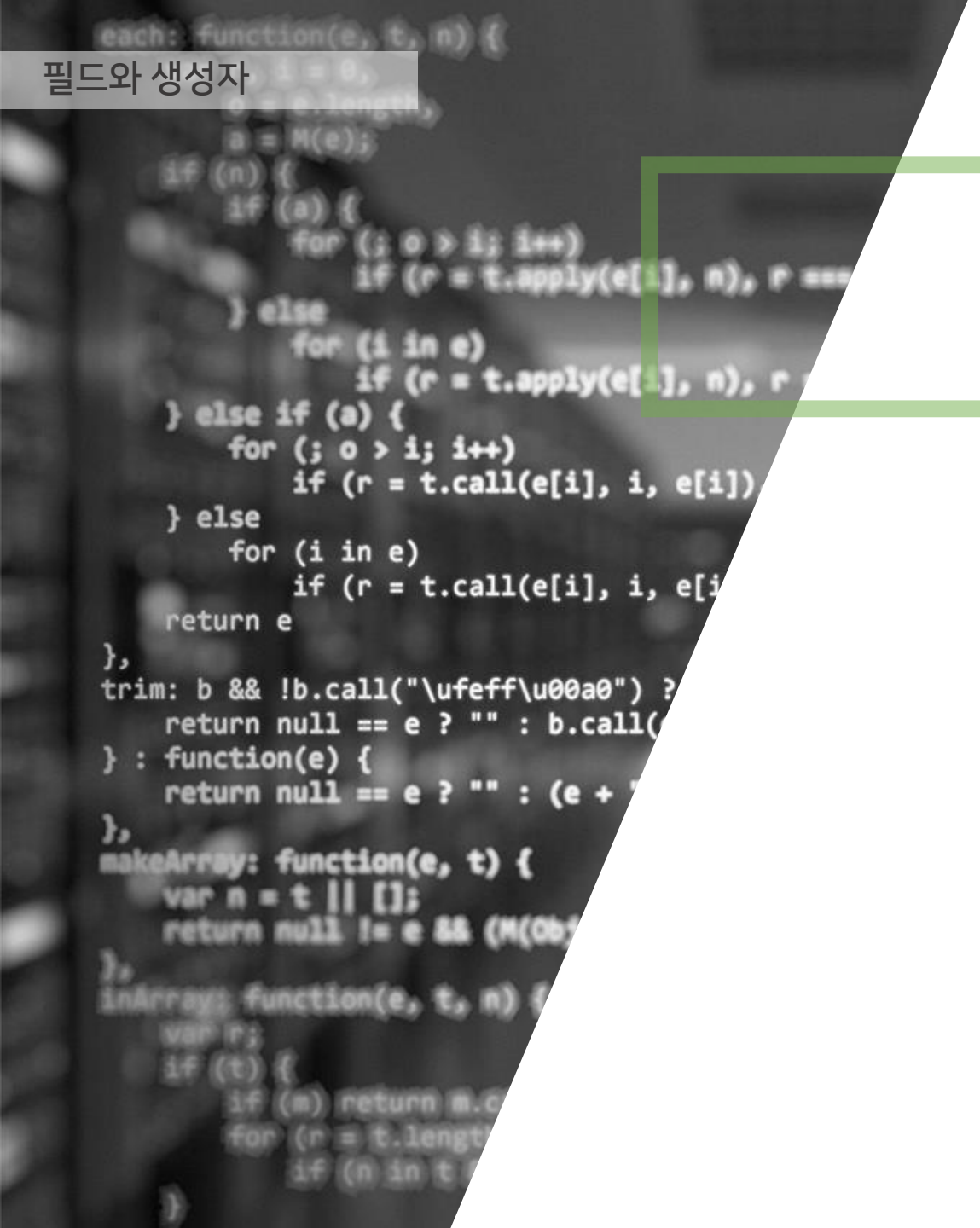
제07장

필드와 생성자

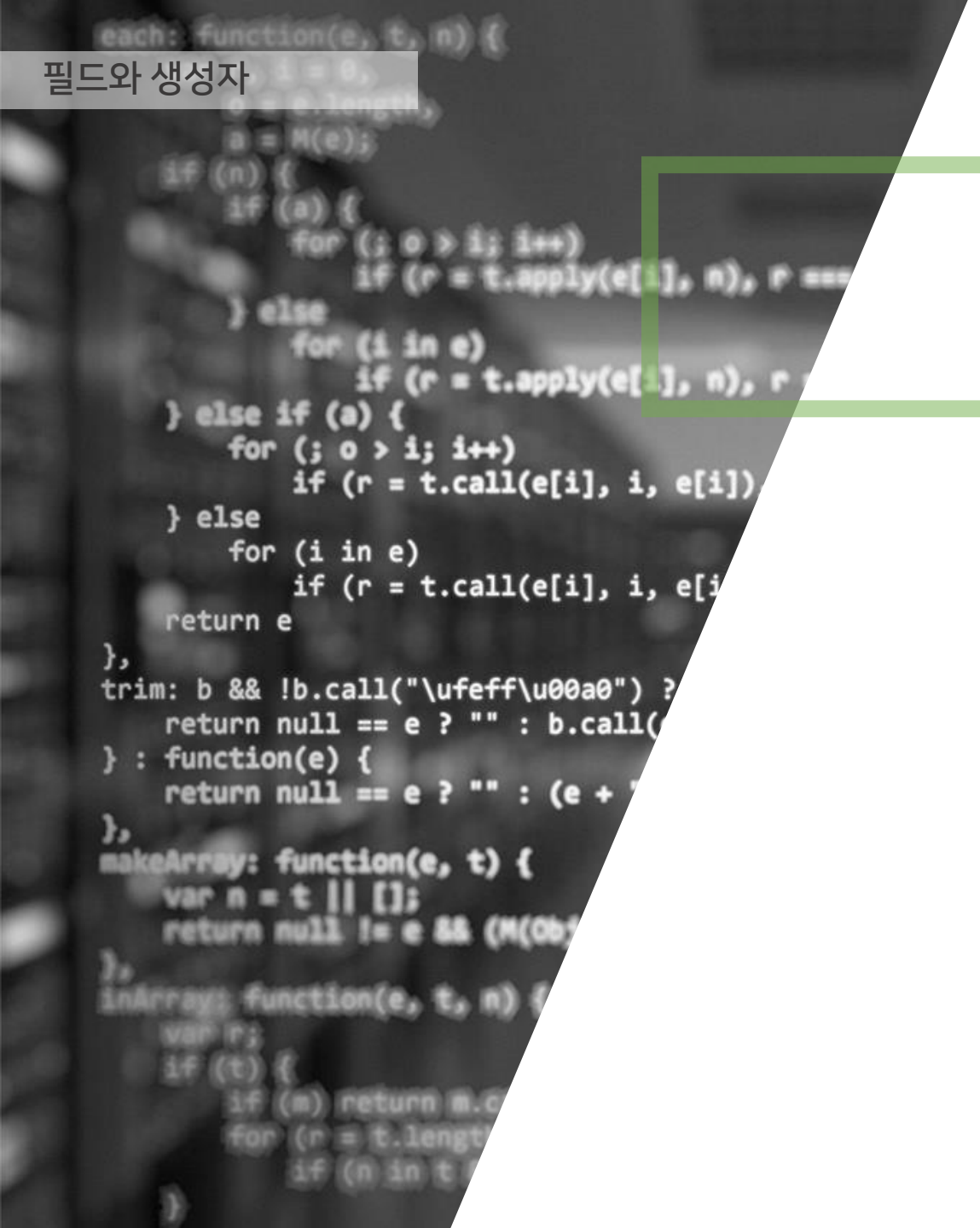
구디아카데미 ▷ 민경태 강사

Java

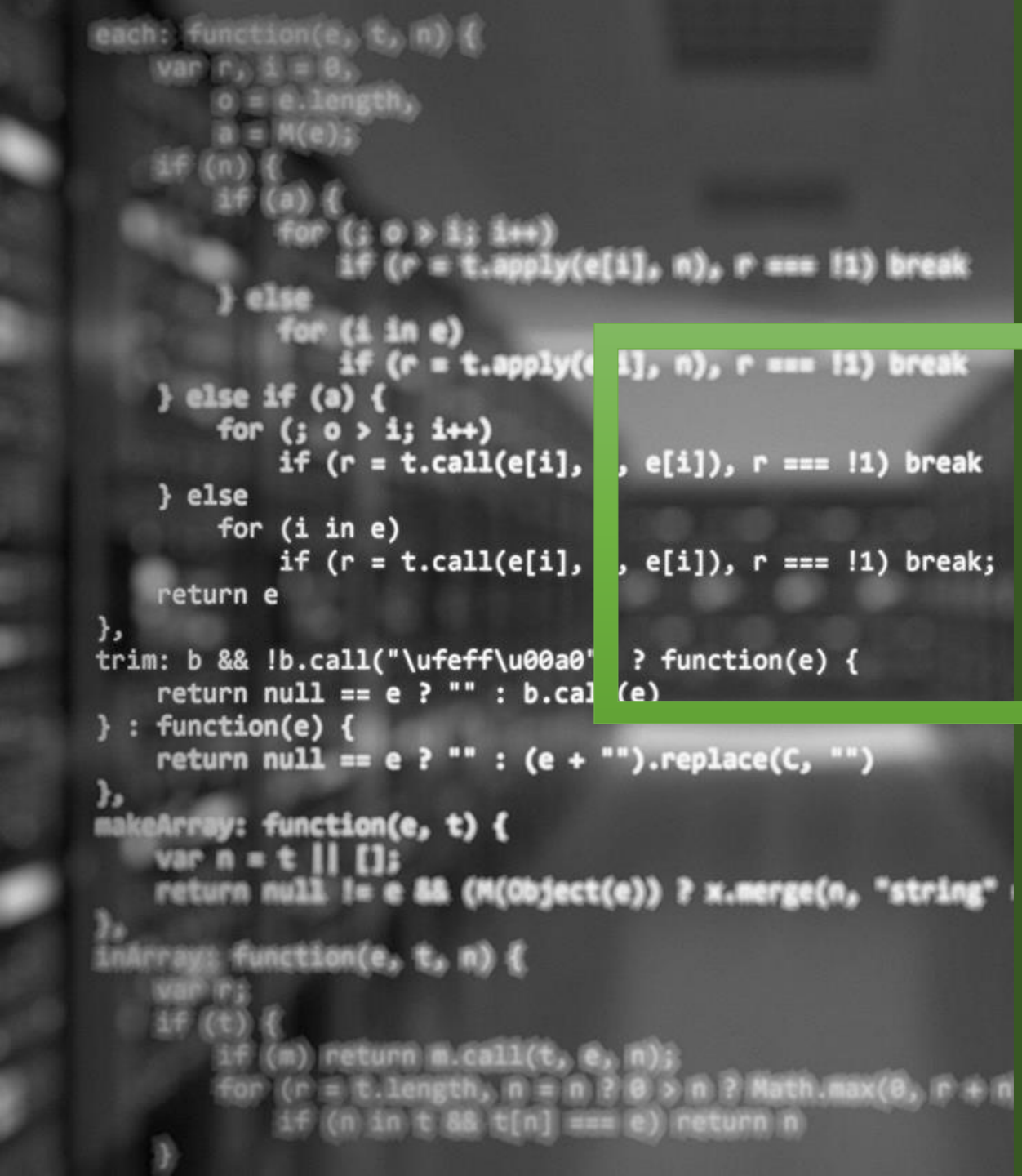


학습목표

1. 필드에 대해서 알 수 있다.
2. this에 대해서 알 수 있다.
3. 생성자에 대해서 알 수 있다.
4. 생성자 오버로딩에 대해서 알 수 있다.



목차


each: function(e, t, n) {
var r, i = 0,
o = e.length,
a = M(e);
if (n) {
if (a) {
for (; o > i; i++)
if (r = t.apply(e[i], n), r === !1) break
} else
for (i in e)
if (r = t.apply(e[i], n), r === !1) break
} else if (a) {
for (; o > i; i++)
if (r = t.call(e[i], e[i]), r === !1) break
} else
for (i in e)
if (r = t.call(e[i], e[i]), r === !1) break;
return e
},
trim: b && !b.call("\ufeff\u00a0") ? function(e) {
return null == e ? "" : b.call(e)
} : function(e) {
return null == e ? "" : (e + "").replace(C, "")
},
makeArray: function(e, t) {
var n = t || [];
return null != e && (M(Object(e)) ? x.merge(n, "string"
},
inArray: function(e, t, n) {
var r;
if (t) {
if (m) return m.call(t, e, n);
for (r = t.length, n = n ? 0 > n ? Math.max(0, r + n
if (n in t && t[n] === e) return n
}

1. 필드

필드

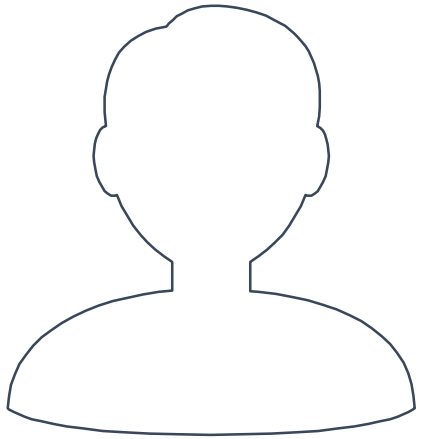
- Field
- 클래스에 선언한 변수를 필드라고 함
- 클래스를 기반으로 생성하는 객체들이 가지는 속성을 저장함
- 필드는 동일한 클래스에 있는 모든 메소드가 사용할 수 있음

나는 팥 붕어빵
(필드 : 팥)

나는 슈크림 붕어빵
(필드 : 슈크림)

필드의 이해

- 객체마다 서로 다른 필드 값을 가질 수 있음



	초기상태	객체1	객체2	객체3	객체4
필드	이름 : null 나이 : 0 직책 : null 결혼유무 : false	이름 : 전지현 나이 : 35 직책 : 팀장 결혼유무 : true	이름 : 소지섭 나이 : 30 직책 : 과장 결혼유무 : false	이름 : 황정민 나이 : 40 직책 : 이사 결혼유무 : true	이름 : 배수지 나이 : 27 직책 : 주임 결혼유무 : false

필드 초기화

- 필드는 선언만 해도 자동으로 초기화 됨

- 필드의 자료형에 따른 자동 초기화 값

유형	자료형	값
기본 자료형	정수형	0
	실수형	0.0
	논리형	false
	문자형	₩u0000 (null문자)
참조 자료형		null

메소드의 필드 접근

- 메소드는 필드에 그냥 접근할 수 있음

```
public class Car {

    String model;

    void info() {
        System.out.println(model);
    }

}
```

필드 선언

메소드에서 필드에 접근

클래스와 객체의 필드 접근

- 클래스의 멤버(필드, 메소드)는 모두 마침표(.)를 이용하여 접근할 수 있음
- 주로 객체를 이용해서 접근하나 클래스 필드의 경우 클래스를 이용해서 접근할 수 있음(*static 필드)
- 형식

  객체.필드

  클래스.필드

* static 키워드는 [08.접근제어자와정적멤버]에서 다룬다.

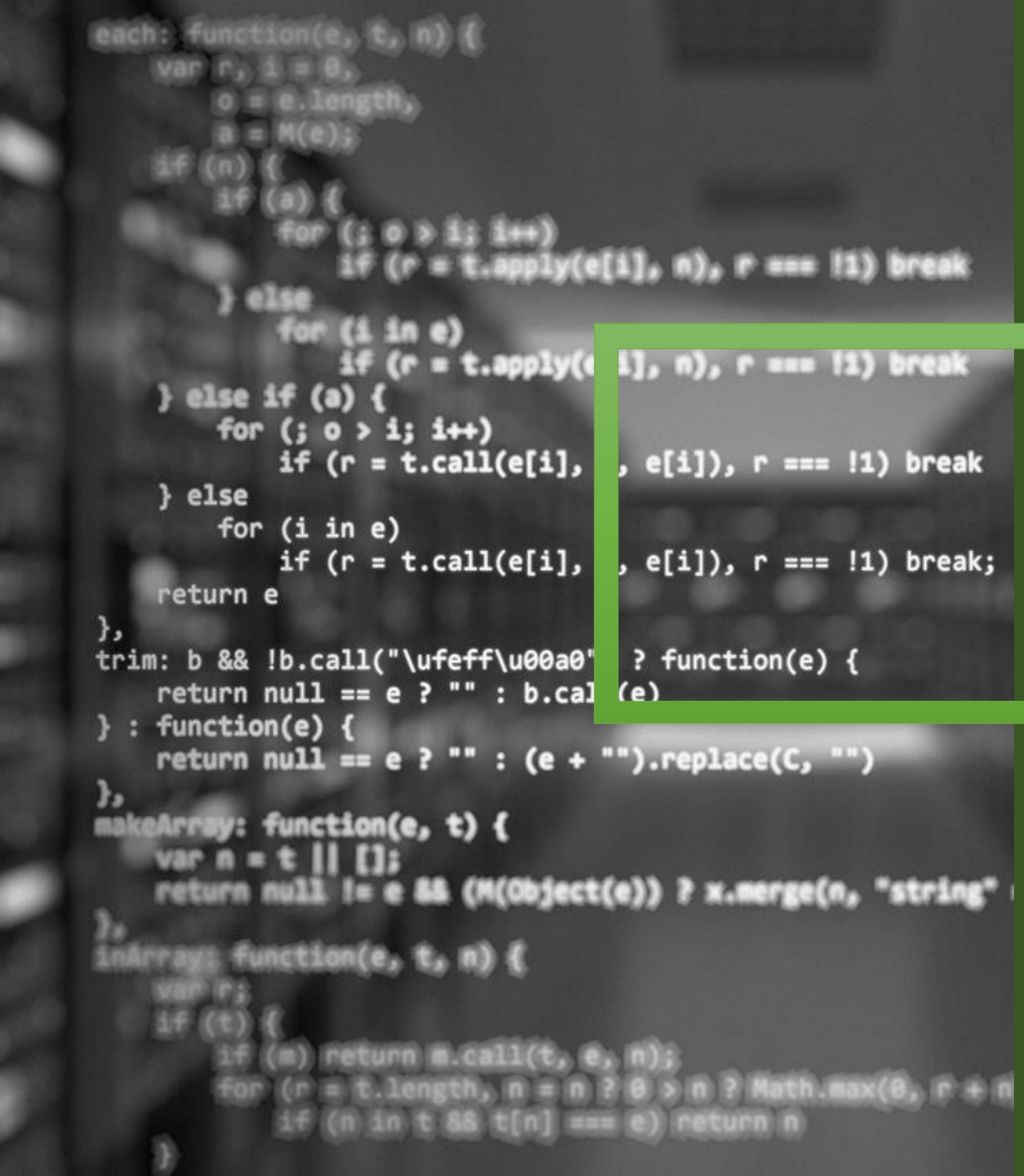
클래스와 객체의 필드 접근 예시

- 객체를 이용한 필드 접근

```
public class Car {
    String model;
}
```

```
public class MainClass {
    public static void main(String[] args) {
        Car car = new Car();
        car.model = "폴리";
        System.out.println( car.model );
    }
}
```

객체.필드 형식으로 접근한다.


each: function(e, t, n) {
var r, i = 0,
o = e.length,
a = M(e);
if (n) {
if (a) {
for (; o > i; i++)
if (r = t.apply(e[i], n), r === !1) break
} else
for (i in e)
if (r = t.apply(e[i], n), r === !1) break
} else if (a) {
for (; o > i; i++)
if (r = t.call(e[i], e[i]), r === !1) break
} else
for (i in e)
if (r = t.call(e[i], e[i]), r === !1) break;
return e
},
trim: b && !b.call("\ufeff\u00a0") ? function(e) {
return null == e ? "" : b.call(e)
} : function(e) {
return null == e ? "" : (e + "").replace(C, "")
},
makeArray: function(e, t) {
var n = t || [];
return null != e && (M(Object(e)) ? x.merge(n, "string"
},
inArray: function(e, t, n) {
var r;
if (t) {
if (m) return m.call(t, e, n);
for (r = t.length, n = n ? 0 > n ? Math.max(0, r + n
if (n in t && t[n] === e) return n
}

2. this

this

- 객체 자신의 참조값을 의미하는 키워드

- 클래스 내부에서만 활용 가능함

- 주요 사용 형식
  - this.필드　　현재 클래스의 필드값을 참조함
  - this.메소드()　현재 클래스의 메소드를 호출함
  - this()　　현재 클래스의 생성자를 호출함

this는 객체의 참조 값

- this는 현재 객체의 참조 값을 의미함

```
public class MyClass {


   void method() {

      System.out.println(this);

   }


}
```

```
public class MainClass {

   public static void main(String[] args)  {

      MyClass myClass = new MyClass();

      System.out.println(myClass);

      myClass.method();

   }

}
```

System.out.println(myClass); → 객체 참조 값 출력

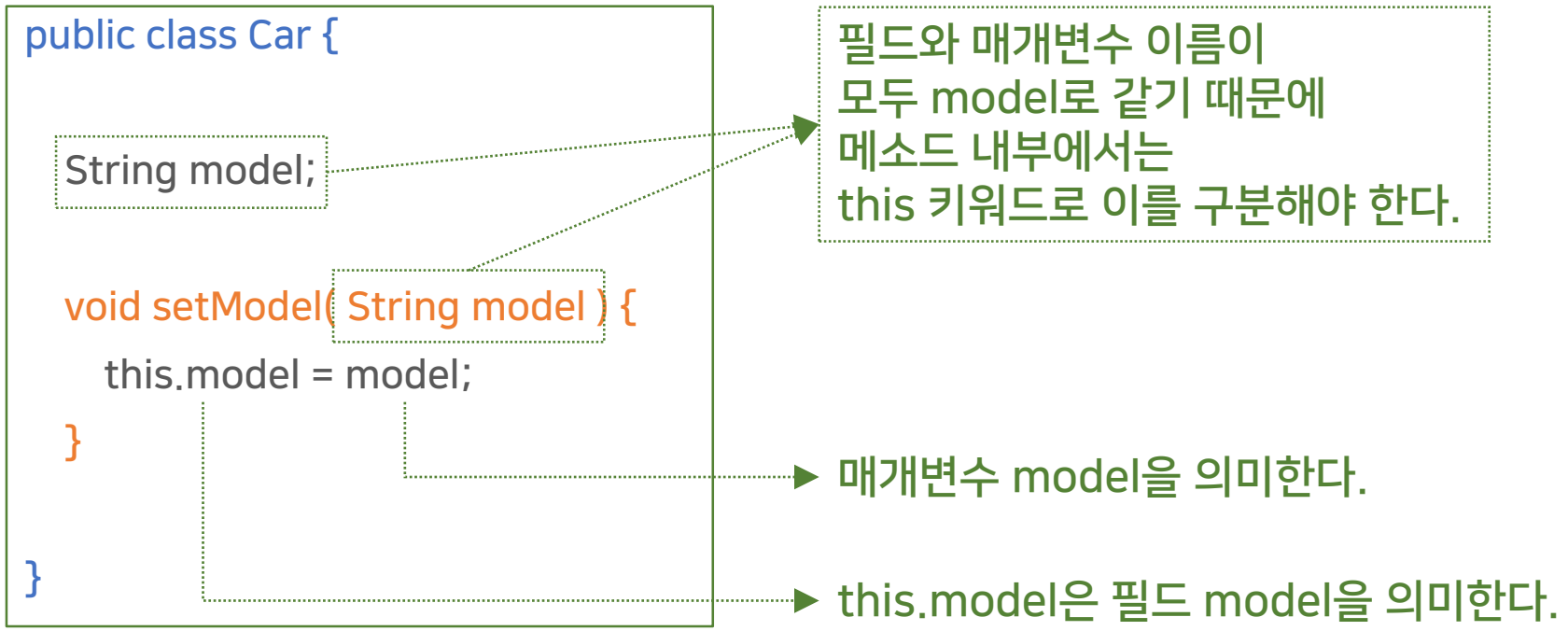
myClass.method(); → this 출력

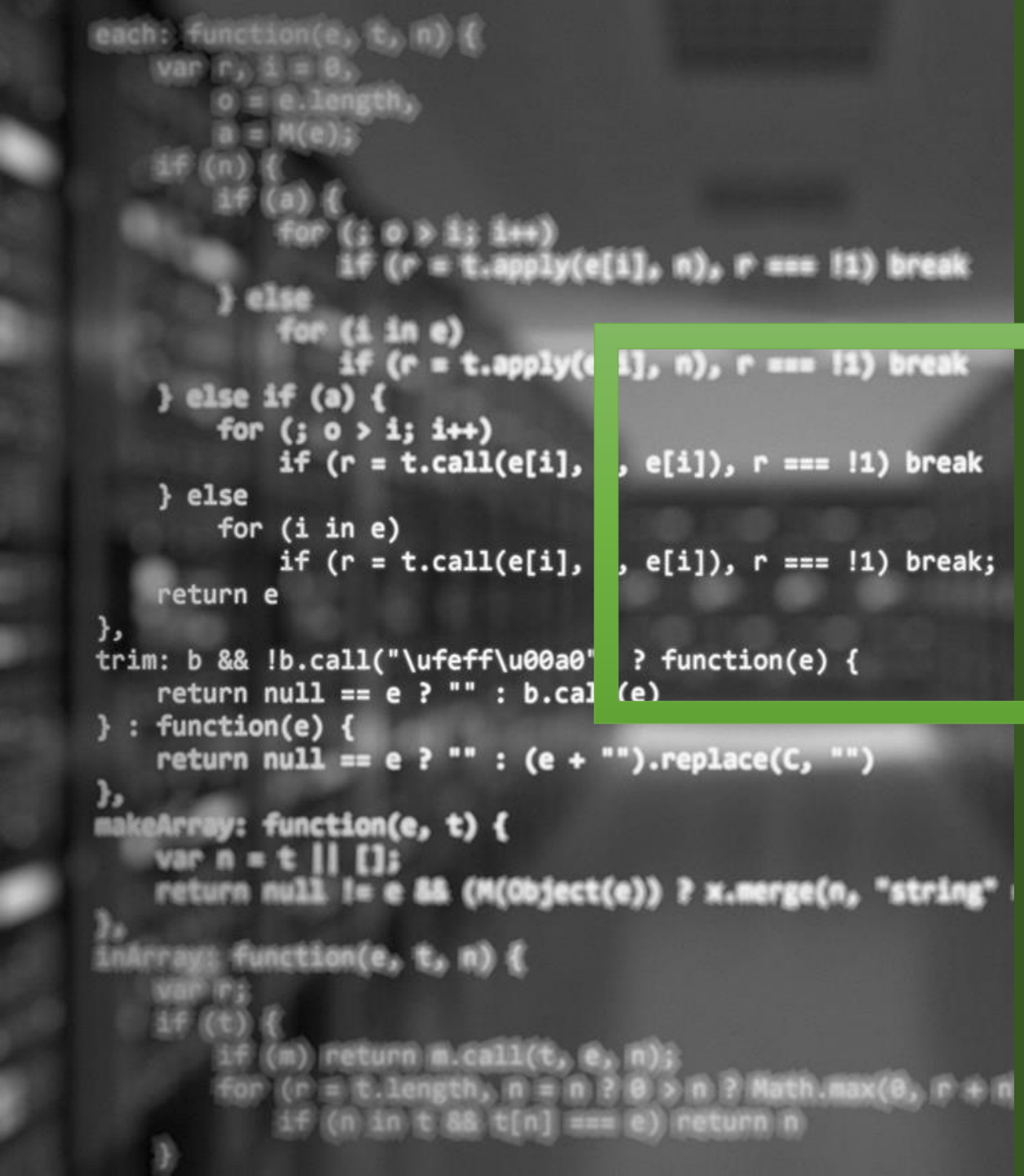
실행결과 (실행 결과는 다를 수 있으나 두 값이 동일하게 나오는지 확인)

```
pkg.MyClass@742a958e
pkg.MyClass@742a958e
```

this 활용

- 필드이름과 매개변수이름이 같을 때 this 키워드를 많이 활용함




each: function(e, t, n) {
var r, i = 0,
o = e.length,
a = M(e);
if (n) {
if (a) {
for (; o > i; i++)
if (r = t.apply(e[i], n), r === !1) break
} else
for (i in e)
if (r = t.apply(e[i], n), r === !1) break
} else if (a) {
for (; o > i; i++)
if (r = t.call(e[i], e[i]), r === !1) break
} else
for (i in e)
if (r = t.call(e[i], e[i]), r === !1) break;
return e
},
trim: b && !b.call("\ufeff\u00a0") ? function(e) {
return null == e ? "" : b.call(e)
} : function(e) {
return null == e ? "" : (e + "").replace(C, "")
},
makeArray: function(e, t) {
var n = t || [];
return null != e && (M(Object(e)) ? x.merge(n, "string"
},
inArray: function(e, t, n) {
var r;
if (t) {
if (m) return m.call(t, e, n);
for (r = t.length, n = n ? 0 > n ? Math.max(0, r + n
if (n in t && t[n] === e) return n
}

3. 생성자

생성자

- Constructor

- 객체를 생성할 때만 호출되는 특별한 메소드

- 객체를 생성하는 시점 이외에는 임의로 호출할 수 없음

- 형식

```
생성자명(매개변수) {
    생성자 본문
}
```

생성자의 특징

- 클래스 이름과 생성자의 이름이 같음
- 반환타입이 존재하지 않음
- 매개변수 작성 방법은 일반 메소드와 동일함
  - 메소드 오버로딩이 가능함

생성자의 모습

- 생성자는 반환타입이 없고 클래스와 이름이 같은 메소드

```
public class Member {

    Member() {

    }

}
```

```
public class Board {

    Board() {

    }

}
```

```
public class Product {

    Product() {

    }

}
```

모두 생성자이다.
▶ 반환타입이 없고,
클래스와 이름이 같다.

...

디폴트 생성자

- Default Constructor

- 생성자가 없는 클래스를 컴파일할 때 자바 컴파일러가 자동으로 생성해서 제공하는 생성자

- 디폴트 생성자는 매개변수와 본문이 없음

- 형식

  생성자명( ) { }

생성자의 호출

- new 키워드 다음 부분이 바로 생성자를 호출하는 부분임

```
public class Car {

    Car() {
        System.out.println("Car 생성");
    }

}
```

생성자

```
public class MainClass {

    public static void main(String[] args) {
        Car car = new Car() ;
    }

}
```

생성자가 호출되는 곳

생성자의 목적

- 일반적으로 생성자는 필드 초기화를 위해서 사용함

```
public class Car {

   String model;

   Car(String model) {
      this.model = model;
   }
}
```

② 매개변수로 전달된 "폴리"를 필드 model에 저장한다.

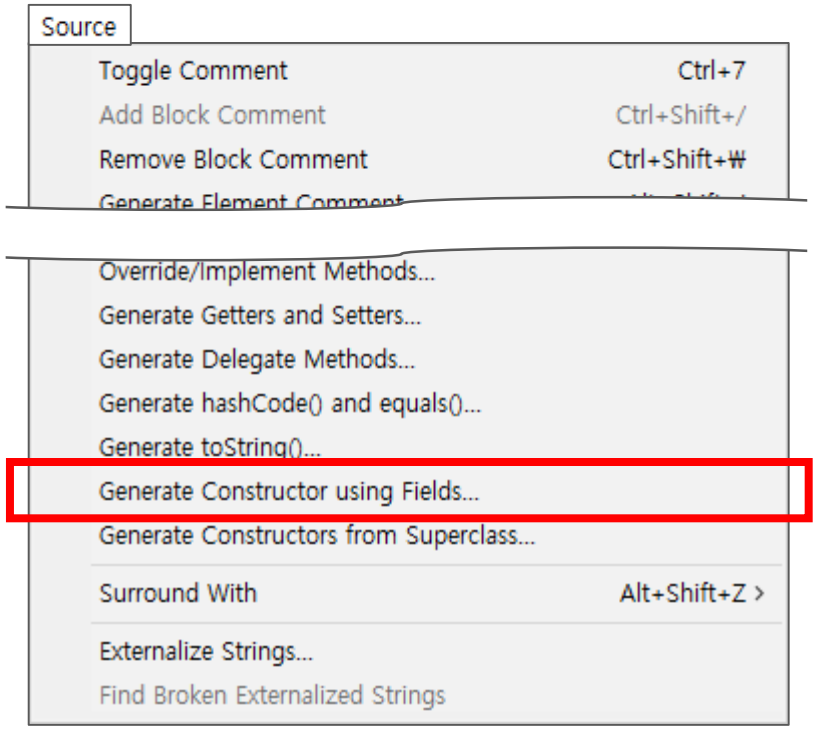
```
public class MainClass {

   public static void main(String[] args) {
      Car car = new Car("폴리") ;
   }
}
```

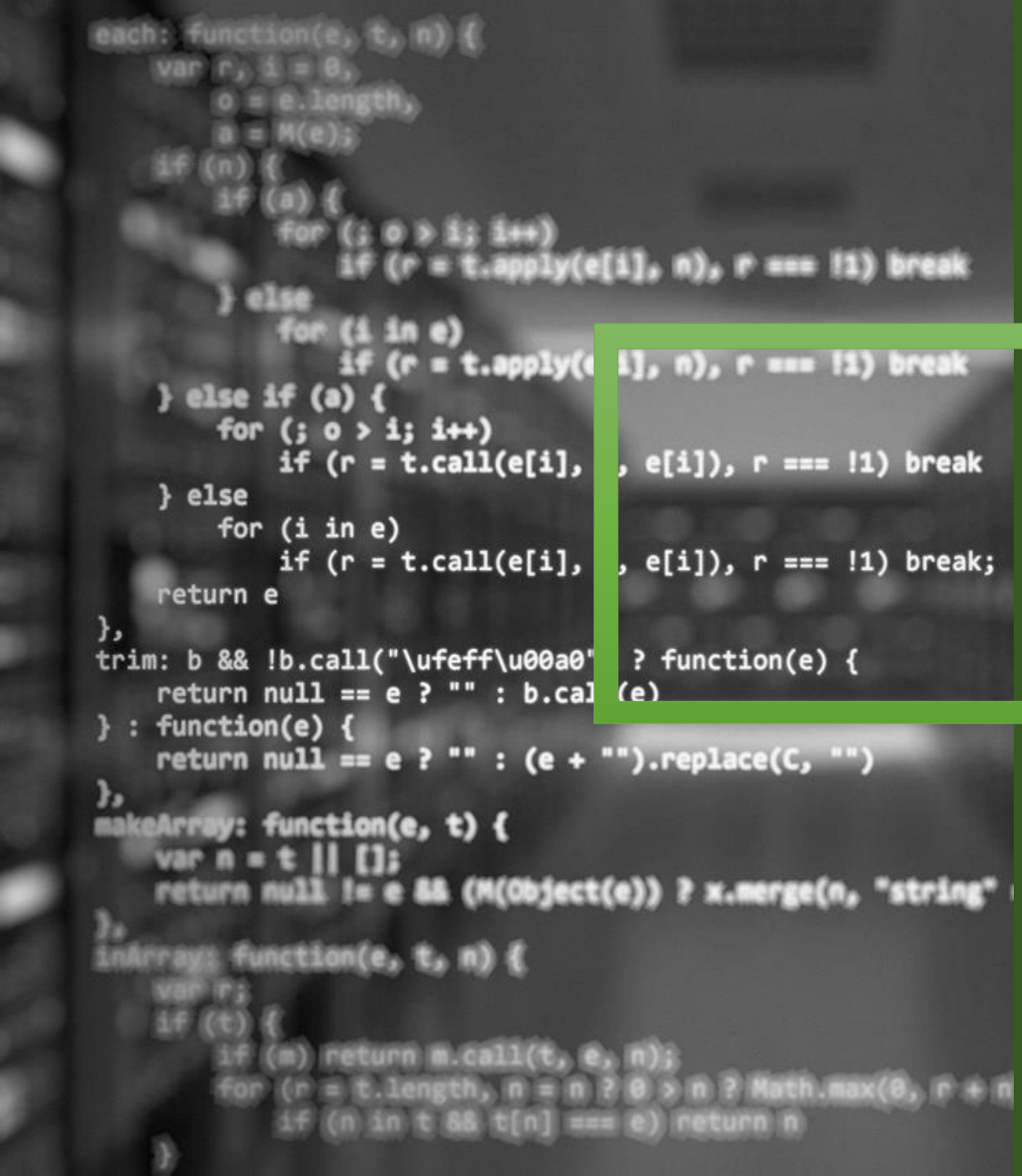
① Car(String model) { } 생성자를 호출하고 "폴리"를 인자로 전달한다.

Generate Constructor using Fields

- 이클립스는 자동으로 생성자를 만들어 줌

- [Source] - [Generate Constructor using Fields...]





4. 생성자 오버로딩

생성자 오버로딩

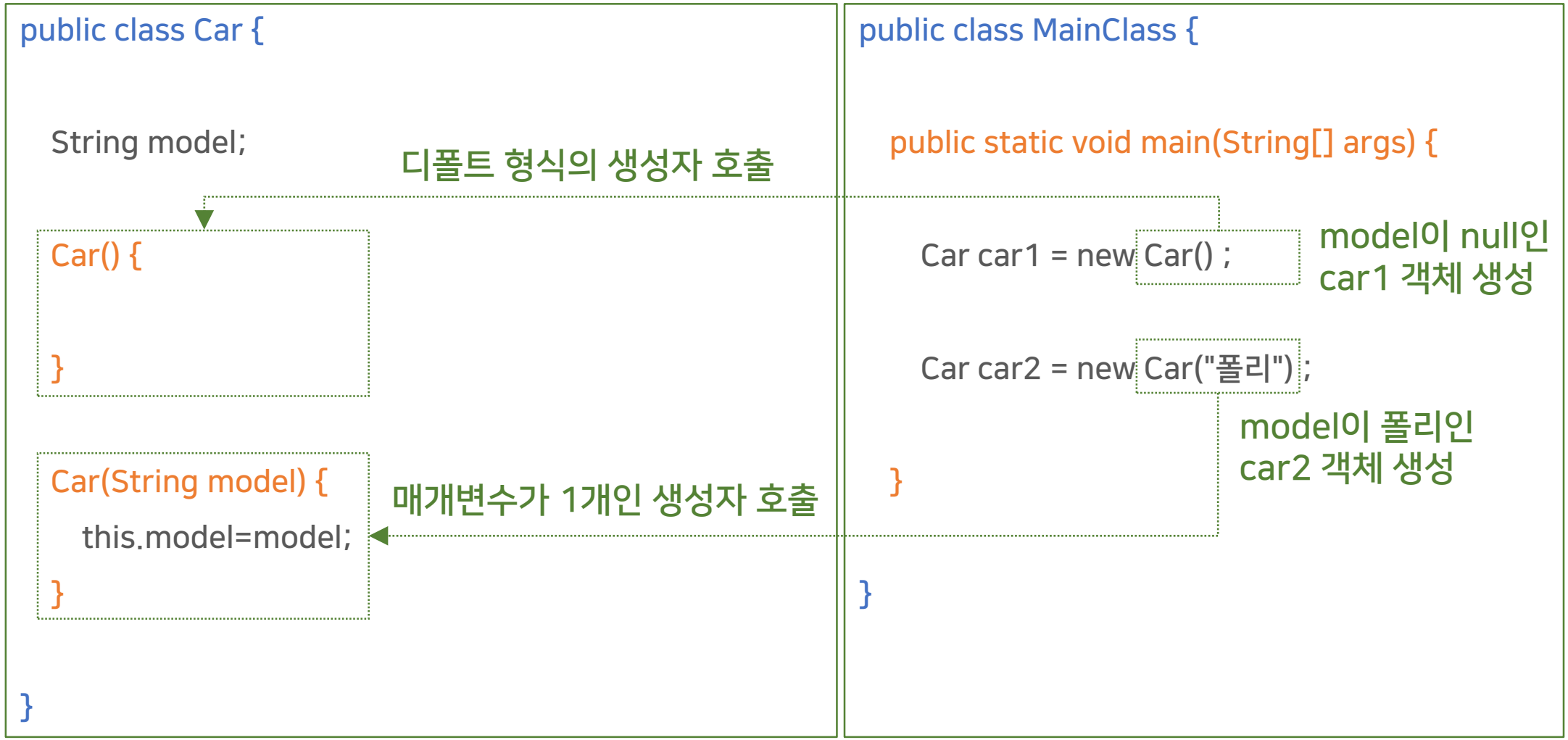
- 생성자는 일반 메소드처럼 메소드 오버로딩이 가능함

- 생성자 오버로딩 규칙(모두 만족해야 함)
  1. 같은 생성자이름
  2. 다른 매개변수

- 생성자 오버로딩은 생성자를 n개 만들 수 있음을 의미하는데 이것은 객체 생성 방법이 n개 라는 것을 의미함

생성자 오버로딩 예시

- 생성자가 2개인 Car 클래스



this()

- 생성자 내부에서 다른 생성자를 호출할 때 this() 사용