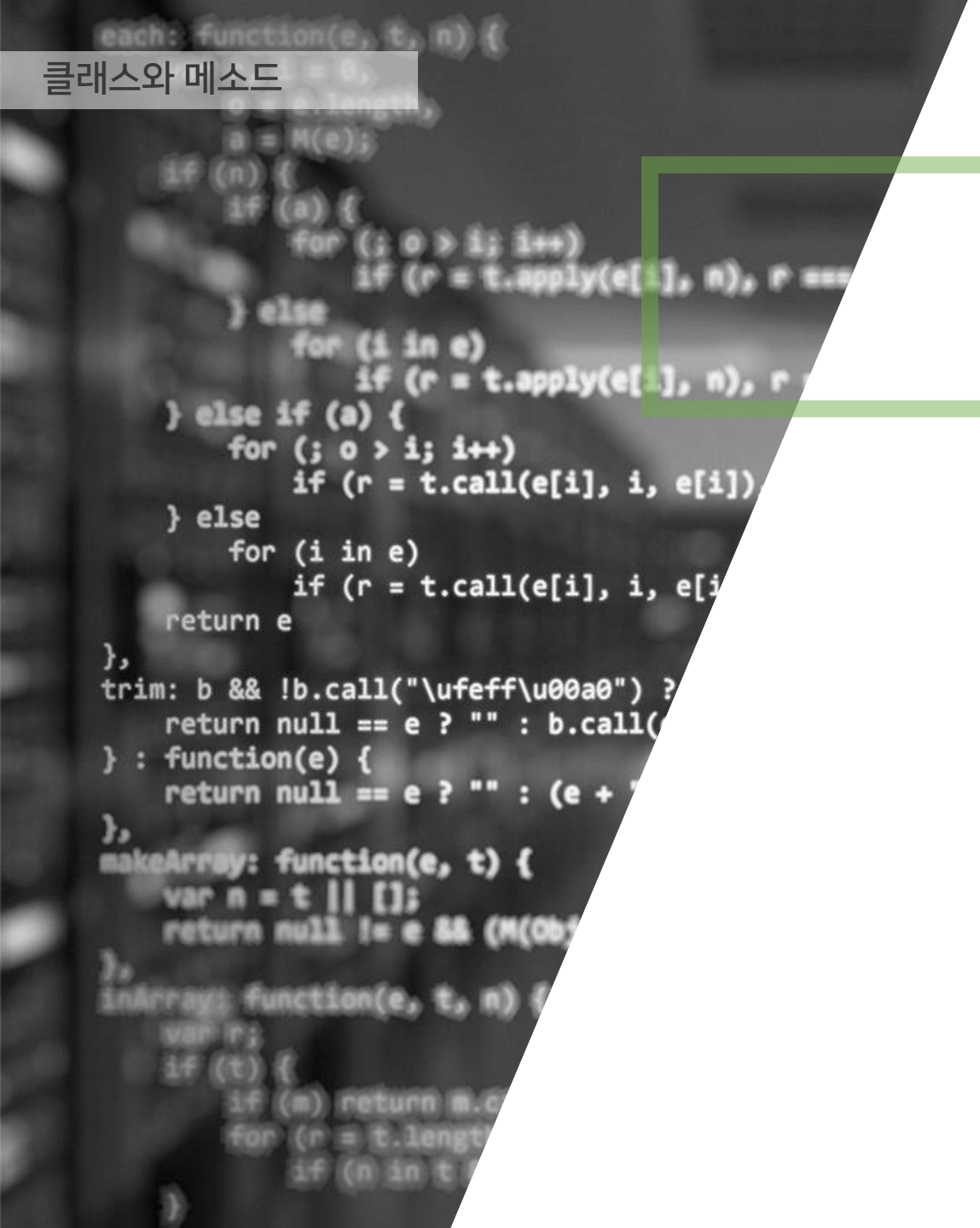
제06장

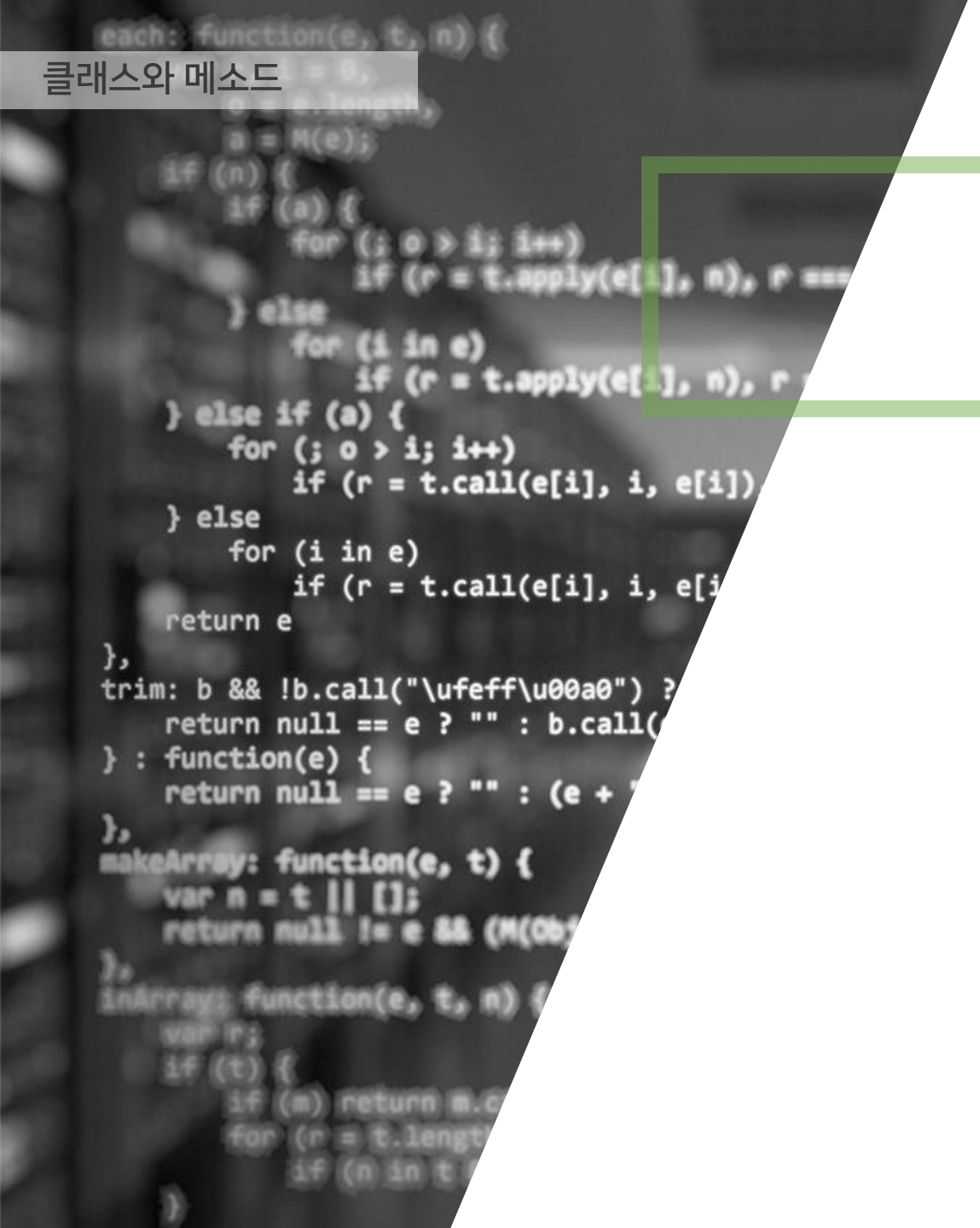
클래스와 메소드

구디아카데미 ▷ 민경태 강사

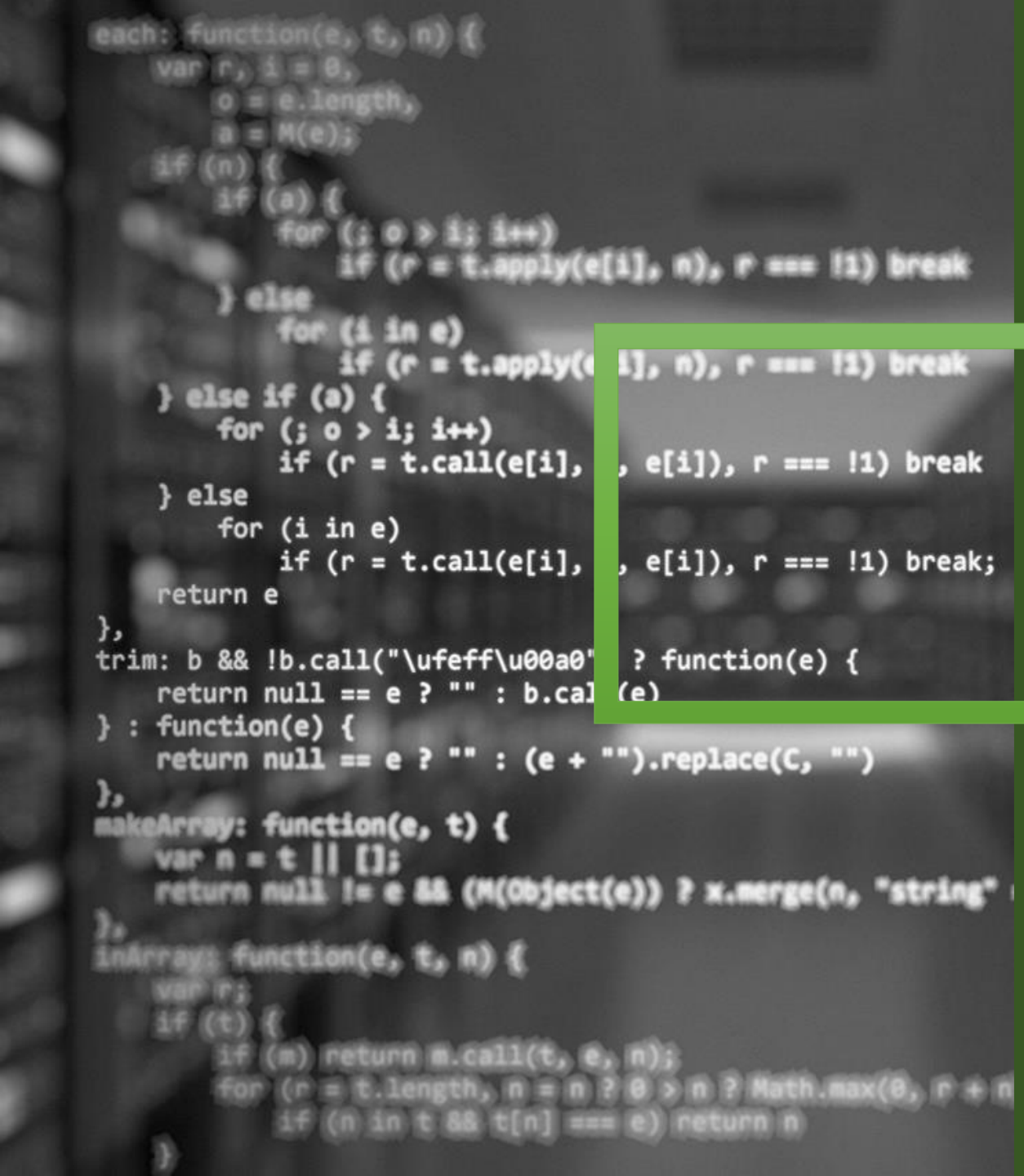


학습목표

1. 클래스와 객체의 개념에 대해서 알 수 있다.
2. 클래스를 만들 수 있다.
3. 생성된 클래스를 기반으로 객체를 만들 수 있다.
4. 메소드를 정의하고 호출할 수 있다.
5. 인자와 매개변수를 처리할 수 있다.
6. 메소드 오버로딩에 대해서 이해할 수 있다.
7. 메소드의 반환값을 처리할 수 있다.



목차



1. 클래스와 객체

객체

- Object
- 클래스를 이용해서 만들어 낼 수 있는 자바의 데이터를 의미함
- 자바의 8가지 기본 타입 변수를 제외한 나머지 데이터는 모두 객체임
- 객체는 참조 타입을 가짐

클래스

- Class

- 객체를 만들기 위한 설계도의 개념

- 자바의 모든 코드는 클래스 내부에 작성해야 함 (import 등 일부 코드 제외)

- 클래스는 필드와 메소드로 구성됨
  - 필드 : 데이터 저장을 위한 변수
  - 메소드 : 기능 구현을 위한 함수

클래스와 객체의 관계

- 클래스와 객체는 일대다 관계(1:N)를 가짐

- 하나의 클래스로 여러 개의 객체를 만들 수 있음

클래스
(붕어빵 설계도)

객체
(붕어빵)

클래스 생성

- 클래스 이름은 Pascal Case 규칙을 따름

- class 키워드 뒤에 생성할 클래스 이름을 작성함

- 형식

```
class 클래스명 {

}
```

이클립스의 클래스 생성

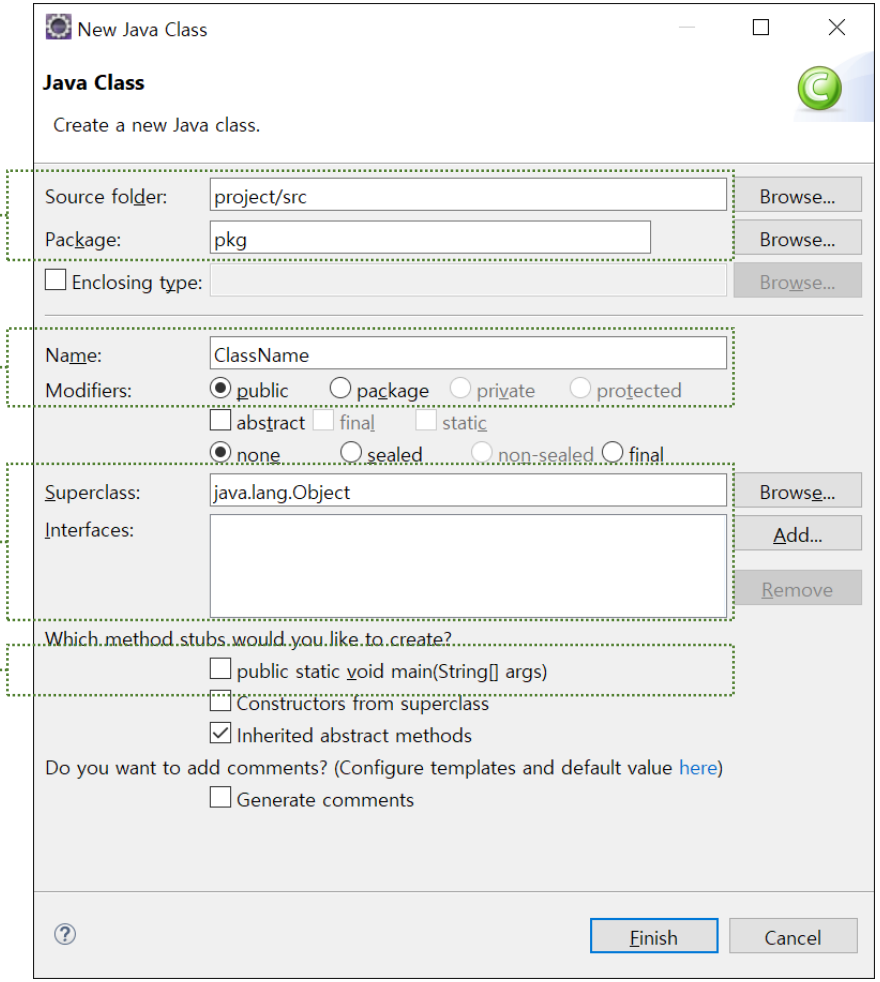
- [File] - [New] - [Class]

프로젝트 / 소스 폴더 / 패키지

클래스 / 접근제어자

슈퍼 클래스 / 인터페이스

메인 메소드



이클립스의 클래스 생성과 자바 파일

- 클래스를 추가하면 클래스와 같은 이름의 자바 파일(*.java)이 만들어짐

- 생성된 자바 파일 내부에 또 다른 클래스를 추가할 수 있으나 자바 파일과 같은 이름을 가진 클래스만 public 키워드*를 가질 수 있음

- 이클립스가 생성하는 클래스 형식

```
public class 클래스명 {

}
```

* public 키워드는 [08.접근제어자와정적멤버]에서 다룬다.

이클립스의 클래스 생성 예시

- 이클립스를 이용한 MyClass 클래스 생성

파일명	MyClass.java	myClass.java	myClass.java
클래스	public class MyClass { }	public class myClass { }	public class MyClass { }
	성공 1. 파일명==클래스명 (O) 2. Pascal Case (O)	경고 (실행됨) 1. 파일명==클래스명 (O) 2. Camel Case (X)	오류 (실행안됨) 1. 파일명!=클래스명 (X) 2. Pascal Case (O)

객체 생성 과정

- 객체를 만들기 위해서는 아래 2가지 작업이 필요함
  - 객체 선언
  - 객체 생성

- 객체 선언과 객체 생성

구분	객체 선언	객체 생성
의미	객체를 참조할 참조 변수를 선언하는 것	실제로 객체를 생성하는 것
메모리	참조 변수는 Stack 영역에 생성됨	객체는 Heap 영역에 생성됨
방법	클래스 타입의 참조 변수를 선언함	new 키워드를 이용해 객체를 생성함
예시 코드	MyClass myClass	new MyClass()

객체 선언

- 생성할 객체의 참조값을 저장할 참조 변수를 선언하는 것을 의미함
- 객체 이름은 Camel Case 규칙을 따름
- 객체 선언 형식
  클래스명 객체명

객체 선언 예시

- Apple 클래스와 apple 객체 선언

```
public class Apple {

}
```

Apple 클래스 정의

```
Apple apple;
```

Apple 클래스 타입의 apple 객체 선언

객체 생성

- Heap 영역에 필요한 공간을 할당 받아서 객체가 만들어짐
- new 키워드를 이용해서 객체를 생성함
- JVM은 객체를 생성할 때 항상 생성자*라고 하는 메소드를 호출함
- 객체 생성 형식
  new 생성자()

* 생성자는 [07.필드와생성자]에서 다룬다.

객체 생성 예시

- Apple 클래스와 apple 객체 선언 및 생성

```
public class Apple {

}
```

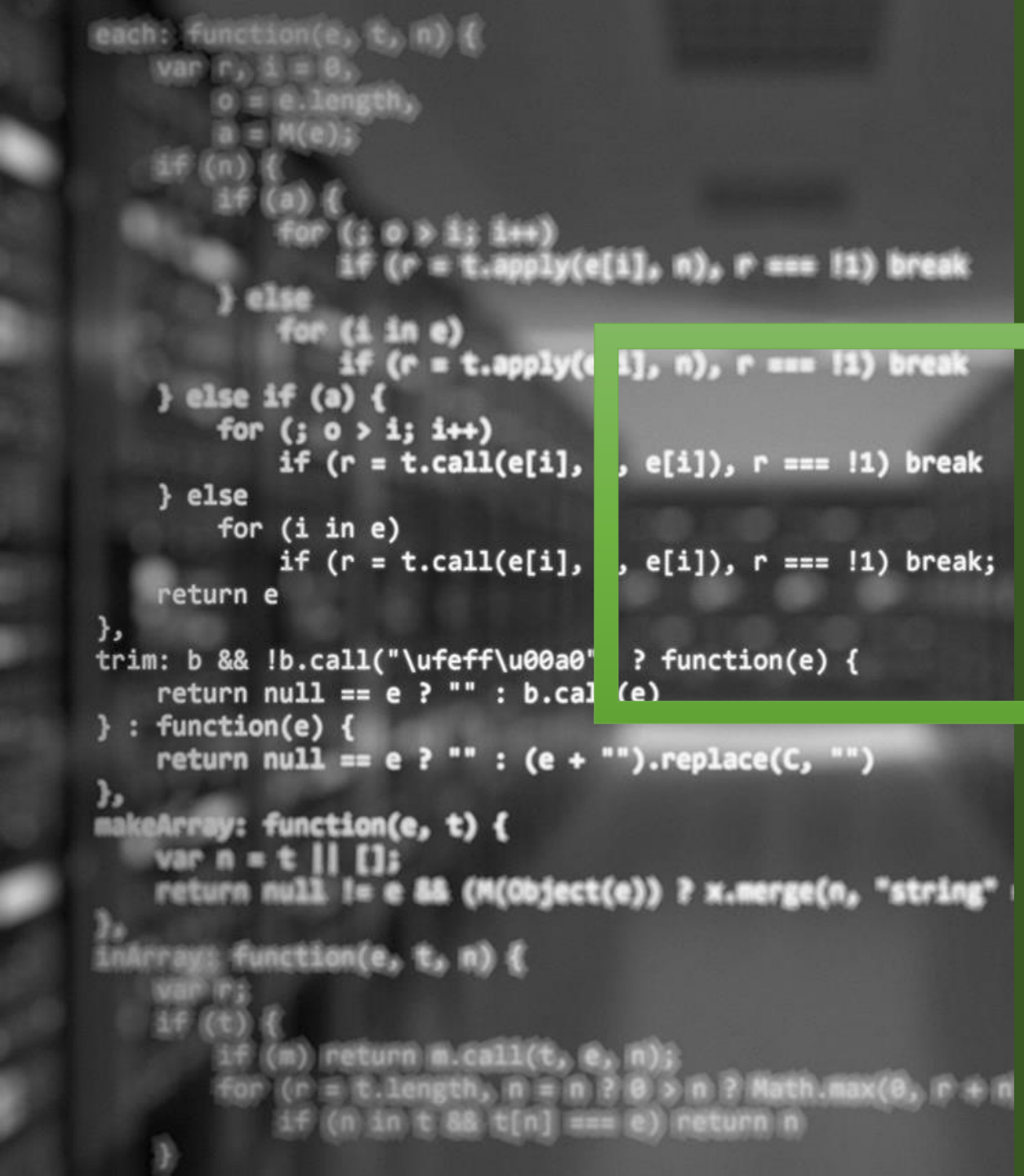
Apple 클래스 정의

```
Apple apple;
```

Apple 클래스 타입의 apple 객체 선언

```
apple = new Apple();
```

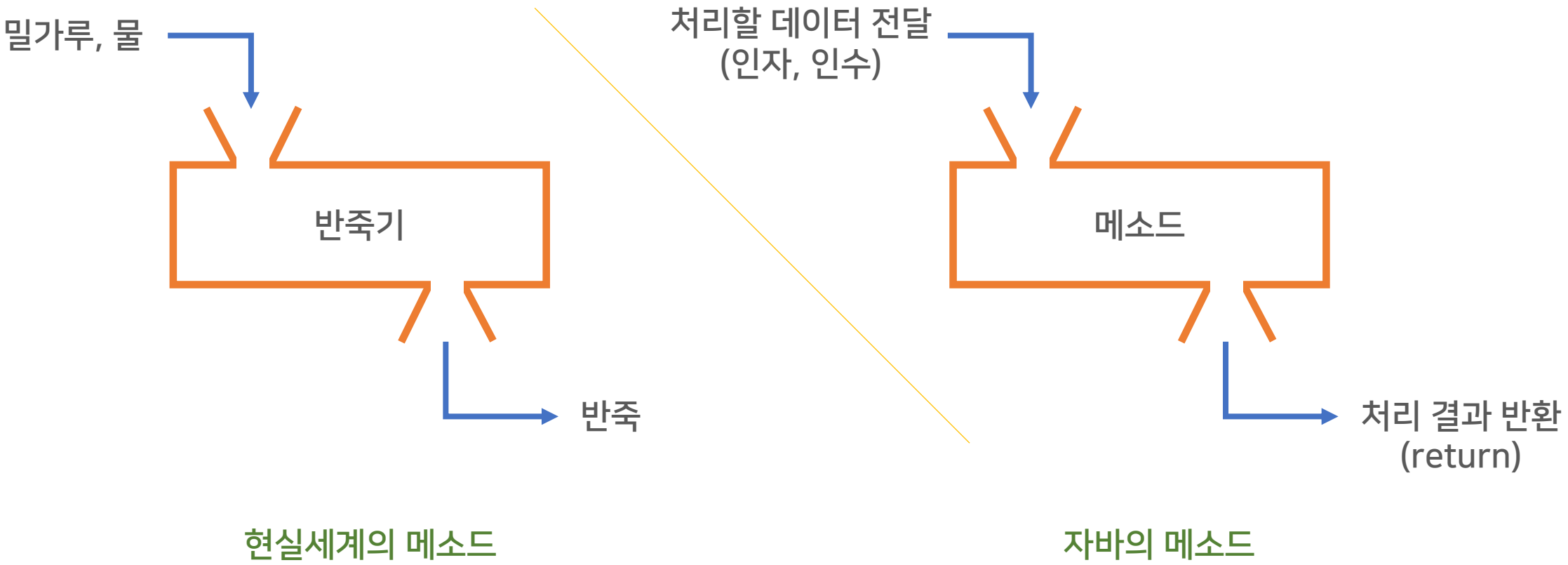
apple 객체 생성


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. 메소드

메소드 개념

- 메소드는 어떤 일을 수행하는 함수를 의미함



메소드

- Method

- 메소드는 객체가 무엇을 할 수 있는지 정의하는 함수를 의미함
  - 함수 : 독립적인 기능을 수행하는 프로그램의 단위
  - 메소드 : 클래스 내부에 작성한 함수

- 클래스 내부에 메소드를 작성해 놓으면 해당 클래스를 이용해서 생성한 객체는 메소드를 사용할 수 있음

메소드 정의

- 메소드 정의 : 클래스 내부에 메소드를 만드는 것을 의미함

- 형식

```
반환타입  메소드명(매개변수) {
    메소드 본문
    [return 반환값]
}
```

- 용어 정리
  - 반환타입 : 반환값(메소드의 실행 결과 값)의 자료형
  - 매개변수 : 메소드 실행시 메소드로 전달되는 값(인자, 인수)을 저장하는 변수
    파라미터(Parameter)라고도 함

메소드 호출

- 메소드 호출 : 클래스 내부에 작성된 메소드를 실행하는 것을 의미함
- 일반적으로 객체를 만든 뒤 객체를 이용해서 메소드를 호출함
- 객체를 만들지 않고 클래스를 이용해서 메소드를 호출하는 경우도 있음
  (*static 메소드)
- 형식
  - 객체.메소드명()
  - 클래스.메소드명()

* static 키워드는 [08.접근제어자와정적멤버]에서 다룬다.

메소드 정의와 호출

- Apple 클래스

```
public class Apple {

    void eat() {
        System.out.println("사과를 먹는다");
    }
                메소드 정의
}
```

```
public class MainClass {

    public static void main(String[] args)  {
        Apple apple = new Apple();
        apple.eat();
        메소드 호출
    }
}
```

실행결과

"사과를 먹는다"

인자와 매개변수

- 인자
  - Argument
  - 인수라고도 함
  - 메소드를 호출할 때 메소드로 전달하는 데이터를 의미함

- 매개변수
  - Parameter
  - 메소드를 정의할 때 메소드로 전달되는 인수를 저장하기 위한 변수를 의미함
  - 매개변수는 메소드 내부에서만 사용할 수 있음

인자를 전달하면 매개변수가 저장한다.

메소드로 값 전달하기

- Calculator 클래스

```
public class Calculator {

    void sum( int a, int b ) {
        System.out.println(a + b);
    }

}
```

매개변수 a, b로 받음

```
public class MainClass {

    public static void main(String[] args)  {
        Calculator calc = new Calculator();
        calc.sum( 10, 20 );
    }

}
```

인자 10, 20 전달

실행결과

```
30
```

메소드로 배열 전달하기

- Calculator 클래스

```
public class Calculator {

    void sum( int[] arr ) {
        int total = 0;
        for(int i = 0; i < arr.length; i++)
            total += arr[i];
        System.out.println(total);
    }

}
```

매개변수 arr로 받음

```
public class MainClass {

    public static void main(String[] args)  {
        Calculator calc = new Calculator();
        calc.sum( new int[] {10, 20, 30} );
    }

}
```

배열 전달

실행결과

```
30
```

가변 인자

- variable argument
- 메소드로 전달하는 인자의 개수가 정해지지 않은 경우에 사용함
- 가변 인자를 처리하는 매개변수는 배열임
- 가변 인자를 처리하는 매개변수는 해당 메소드의 모든 매개변수 중에서 가장 마지막에 선언해야 함
- 가변 매개변수 형식

  반환타입 메소드명(자료형... 매개변수) { }

가변 인자 사용하기

- Calculator 클래스

```
public class Calculator {

    void sum( int... args ) {
        int total = 0;
        for(int i = 0; i < args.length; i++)
            total += args[i];
        System.out.println(total);
    }

}
```

가변 매개변수 args로 받음

```
public class MainClass {

    public static void main(String[] args)  {
        Calculator calc = new Calculator();
        calc.sum( 10, 20 );
        calc.sum( 10, 20, 30 );
    }

}
```

인자 2개 전달

인자 3개 전달

실행결과

```
30
60
```

메소드 오버로딩

- Method Overloading

- 하나의 클래스에 이름이 같은 메소드를 2개 이상 만들 수 있는데 이를 메소드 오버로딩이라고 함

- 메소드 이름이 같은 대신 반드시 매개변수가 다르게 지정되어야만 함
  - 매개변수의 개수가 다르거나 타입이 다르게 지정되어야만 함

- 반환타입은 메소드 오버로딩과 전혀 관련이 없음

메소드 오버로딩 이해하기

- 메소드 오버로딩 예시

```
void aa() {

}
void aa(String str) {

}
```

메소드명 : 일치 매개변수 : 불일치	오버로딩 가능
반환타입 : 일치	관련 없음

```
void bb() {

}
int bb() {

}
```

메소드명 : 일치 매개변수 : 일치	오버로딩 불가능
반환타입 : 불일치	관련 없음

메소드 오버로딩 사용하기

- Calculator 클래스

```
public class Calculator {

    void sum( int a, int b ) {
        System.out.println(a + b);
    }
    void sum( int a, int b, int c ) {
        System.out.println(a + b + c);
    }

}
```

매개변수 a, b로 받음

매개변수 a, b, c로 받음

```
public class MainClass {

    public static void main(String[] args)  {
        Calculator calc = new Calculator();
        calc.sum( 10, 20 );
        calc.sum( 10, 20, 30 );
    }

}
```

인자 2개 전달

인자 3개 전달

실행결과

```
30
60
```

반환값

- 메소드의 실행 결과 값을 의미함
- 메소드의 반환타입과 실제 반환값의 타입이 일치해야 함
- 문법상 하나의 데이터만 반환할 수 있음
  - 여러 개의 값을 반환하려면 모든 값을 하나의 데이터(배열, 객체, 컬렉션 등)로 만들어서 반환해야 함
- 형식

  return 반환값

반환타입

- 반환값의 타입을 의미함
- 메소드를 정의할 때 가장 먼저 나타나는 구문임
- 반환값의 여부에 따른 반환타입
  - 반환값이 없는 경우 : void 키워드를 사용함
  - 반환값이 있는 경우 : 반환값의 자료형을 사용함

반환 위치

- 반환값은 메소드를 호출한 곳으로 값을 반환함

```
public class Calculator {

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

}
```

```
public class MainClass {
                ① 인자 10, 20을 매개변수 int a, int b로 보낸다.

    public static void main(String[] args)  {
        Calculator calc = new Calculator();
        System.out.println( calc.sum( 10, 20 ) );
    }

                ② a+b를 받아온다.
}
```

실행결과

```
30
```

void 반환타입과 return

- void 반환타입의 메소드도 return을 사용할 수 있음
- void 반환타입의 메소드에서는 메소드의 실행을 중지하고 싶을 때 return 키워드를 활용할 수 있음
- 오직 return 키워드만 있고 반환값은 없는 형식으로 구성해야 함
- 형식

```
void 메소드명(매개변수) {
    return;
}
```

메소드 종료를 위한 return

- if문을 이용해 메소드 종료 조건을 만드는 경우가 일반적임

```
void 게임하기() {
    if(숙제제출 == false) {
        return;
    }
    액션게임;
    축구게임;
    보드게임;
}
```

숙제를 제출하지 않았다면 게임하기() 메소드 실행을 종료하시오.

메소드 종료를 위한 return 주의사항

- 반환값이 있는 경우에는 return만으로는 메소드를 종료할 수 없음

```
int 게임하기() {
    if(숙제제출 == false) {
        return 0;
    }
    액션게임;
    축구게임;
    보드게임;
    return 플레이타임;
}
```

숙제를 제출하지 않았다면 0을 반환하시오.

숙제를 제출했다면 플레이타임을 반환하시오.

메소드 종료를 위한 return 예시

- 전달된 점수가 0~100 범위를 벗어난 경우 메소드를 종료함

```
public class Calculator {
   void grade( int score ) {
      if(score < 0 || score > 100) {
         System.out.println("잘못된 점수");
         return;
      }
      String grade = score >= 90 ? "A" :
                           score >= 80 ? "B" :
                           score >= 70 ? "C" :
                           score >= 60 ? "D" : "F";
      System.out.println(score + ":" + grade);
   }
}
```

```
public class MainClass {

   public static void main(String[] args)  {
      Calculator calc = new Calculator();
      calc.grade( 90 );
      calc.grade( 80 );
      calc.grade( 70 );
      calc.grade( 60 );
      calc.grade( 50 );
      calc.grade( -5 );
   }

}
```