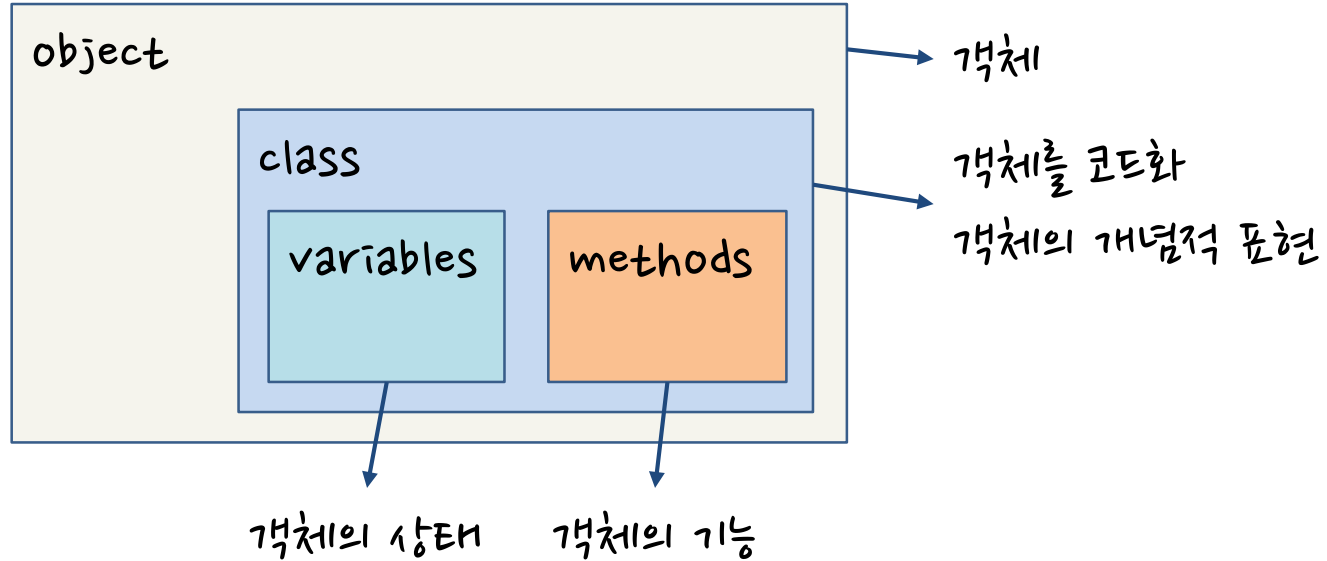


컴퓨터 프로그래밍

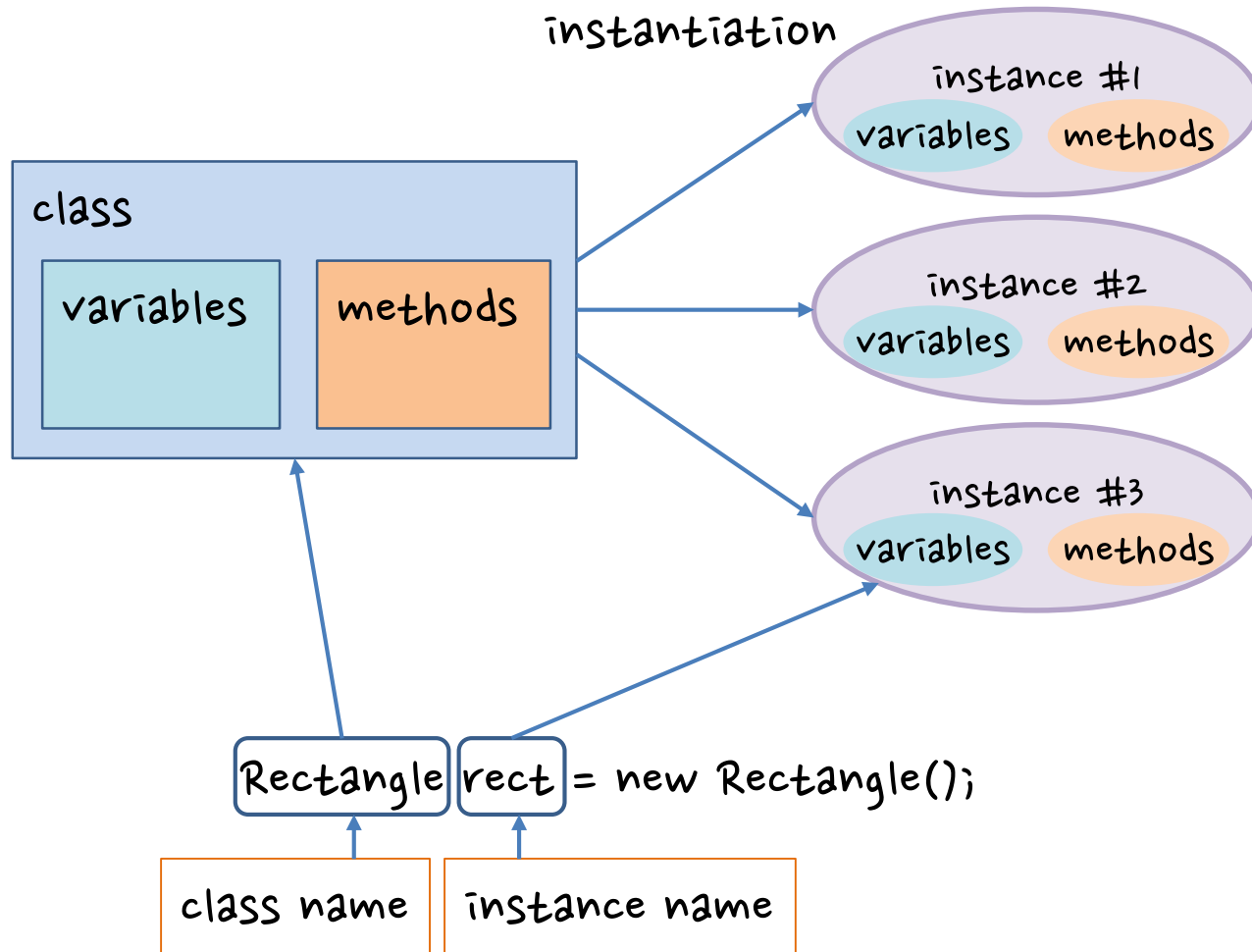
class Review
Method overloading

class Review



class Review

- class를 통해 개별 객체(instance)를 만든다



variables in class

- 클래스 멤버 변수들은 객체의 상태를 표현
- OOP에서는 객체의 상태를 숨기고자 하는 성향이 강하다
— private
- 객체의 상태를 변경하기 위해 public 메소드 제공

```
class AAA
{
    private int num;
    public void setNum(int n) { num=n; }
    public int getNum() { return num; }
    . . . .
}
```

Methods in class

- 객체의 행위, 기능을 표현
- 클래스 멤버 메소드는 클래스 멤버 변수/메소드에 마음대로 접근 가능
 - 많은 메소드들은 자기 자신의 상태(변수)를 변경하는 일을 한다

```
public Rectangle() {  
    x1 = y1 = 0;  
    x2 = y2 = 0;  
    height = calcHeight();  
}
```

```
public int calcHeight() {  
    return Math.abs(y1-y2);  
}
```

class Structure

```
class className {
```

```
    private int intVal;  
    private String strVal;
```

```
    public className() { }  
    public void setIntVal(int i) { intVal = i; }  
    public int getIntVal() { return intVal; }  
    public void setStrVal(String s) { strVal = s; }  
    public String getStringVal() { return strVal; }
```

```
}
```

constructor

- 클래스 멤버 메소드 중 하나
- 객체를 인스턴스화하는 과정에서 단 한번만 실행
- 클래스 멤버 변수의 초기화 수행

```
class className {
```

```
    private int intval;  
    private String strval;
```

```
    public className() { }
```

```
    public void setIntval(int i) { intval = i; }
```

```
    public int getIntval() { return intval; }
```

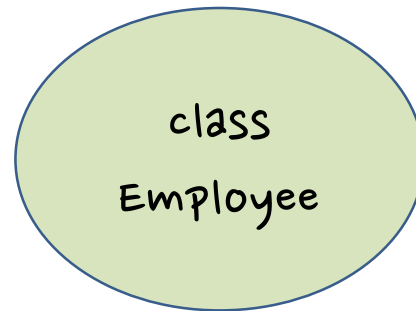
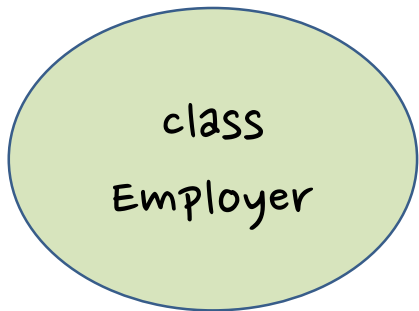
```
    public void setStrval(String s) { strval = s; }
```

```
    public String getStringval() { return strval; }
```

```
}
```

main Method

- class를 만든다고 프로그램이 실행되는 것은 아니다
- Program은 main method로부터 시작된다
- class를 통해 객체를 정의했다면,
main method에서 객체가 해야 할 일을 정의하라



- Employer 인스턴스를 만들어라
- Employee 인스턴스를 만들어라
- Employer로 하여금 Employee에게 임금을 지급하도록 하라

main Method

- main도 클래스 내부에 포함되어야 한다
- 어느 클래스에? don't care
 - 특정 클래스의 내부 혹은 main 메소드만을 가지는 클래스

```
class Employer    /* 고용주 */
{
    private int myMoney;
    public Employer(int money)
    {
        myMoney=money;
    }
    public void payForWork(Employee emp, int money)
    {
        if(myMoney<money)
            return;
        emp.earnMoney(money);
        myMoney-=money;
    }
    public void showMyMoney()
    {
        System.out.println(myMoney);
    }
}
```

java Employer

```
class Employee    /* 고용인 */
{
    private int myMoney;
    public Employee(int money)
    {
        myMoney=money;
    }
    public void earnMoney(int money)
    {
        myMoney+=money;
    }
    public void showMyMoney()
    {
        System.out.println(myMoney);
    }
}
```

↑ java Employee

```
Employer emr=new Employer(3000);
Employee eme=new Employee(0);

emr.payForWork(eme, 1000);
emr.showMyMoney();
eme.showMyMoney();
```

main

Method overloading

- 동일한 이름의 메소드를 둘 이상 정의
- 메소드의 매개변수 개수 또는 자료형이 다르다면 오버로딩 성립
- 오버로딩 된 메소드는 호출 시 전달하는 인자를 통해 구분

```
class Adder {  
    public static int add(int i1, int i2) { ... }  
    public static complex add(complex c1, complex c2) { ... }  
}
```

```
intAddResult = Adder.add(3, 4);
```

```
complexAddResult = Adder.add(new complex(2, 3), new complex(4, 2));
```

constructor overloading

- 생성자 역시 오버로딩 가능

```
class complex {  
    private int real;  
    private int imaginary;  
  
    public complex(int r, int i) {  
        real = r;  
        imaginary = i;  
    }  
    public complex(int r) {  
        real = r;  
        imaginary = 0;  
    }  
}
```

실수와 허수가 모두 주어지는 경우를 위한 생성자

실수만 주어지는 경우를 위한 생성자

this

- 인스턴스 자신을 의미하는 키워드

```
class complex {  
    private int real;  
    private int imaginary;
```

```
    public complex(int r, int i) {  
        real = r;  
        imaginary = i;  
    }
```

```
    public complex(int r) {  
        real = r;  
        imaginary = 0;  
    }
```

```
}
```

```
class complex {  
    private int real;  
    private int imaginary;
```

```
    public complex(int real, int imaginary) {  
        this.real = real;  
        this.imaginary = imaginary;  
    }
```

```
    public complex(int real) {  
        this.real = real;  
        imaginary = 0;  
    }
```

```
}
```

return this

- 인스턴스 자신의 참조 값(this)을 반환
- 연이은 메소드 호출 가능

```
class SimpleAdder {  
    private int num;
```

```
    public SimpleAdder() {  
        num = 0;  
    }
```

```
    public SimpleAdder add(int num) {  
        this.num += num;  
        return this;  
    }
```

```
}
```

```
    public static void main() {  
        SimpleAdder adder = new SimpleAdder();  
        adder.add(1).add(3).add(5);  
    }
```

시작

- 복소수를 표현하기 위한 complex class를 구현
- 덧셈을 위한 add method를 구현
 - add method는 다음의 모든 형태를 계산 가능하도록 구현
 - 복소수 + 복소수
 - 복소수 + 정수
- 연속적인 add method call이 가능하도록 구현
 - e.g.
c.add(1, 2).add(3).add(3, 4);