

05 객체지향과 클래스

Sangwook Lee

05 객체지향과 클래스

1. 객체지향 프로그래밍의 이해

2. 클래스와 객체의 생성

3. 생성자

4. static과 final

1. 객체지향 프로그래밍의 이해

- 객체지향 프로그래밍

컴퓨터 프로그램을
독립된 객체들의 모임으로 보고,
객체를 기반으로
프로그램을 작성하는 방식

1. 객체지향 프로그래밍의 이해

- 절차적 프로그래밍 vs 객체지향 프로그래밍
 - 절차적(procedural) 프로그래밍
 - 함수 단위 프로그래밍
 - 함수 구성: 데이터 정의 부분 + 데이터 처리 부분
 - 객체지향(object-oriented) 프로그래밍
 - 객체 단위 프로그래밍
 - 객체 구성: 필드 + 메소드

1. 객체지향 프로그래밍의 이해

- 필드와 메소드

- 필드(field)

- 객체의 상태(state) 저장 부분
 - 변수(variable)로 구현

- 메소드(method)

- 객체의 행동(behavior) 표현 부분
 - 함수(function)로 구현

= 속성(properties)

= 기능(function)

1. 객체지향 프로그래밍의 이해

- 객체 vs 클래스 vs 인스턴스

- 객체(object) = *사물(thing)*

- 현실 세계 *객체*

- 상태 + 행동

- 클래스(class)

- 객체를 *추상화*하여 만든 프로그램 코드 템플릿

- 필드 + 메소드

- 인스턴스(instance)

- 프로그램 세계 *객체*

- 클래스를 통해 생성

1. 객체지향 프로그래밍의 이해

- 추상화(abstraction)

- 현실 세계의 사실에서 주어진 문제의 중요한 측면을 주목하여 설명하는 방식
- 현실 세계의 객체에서 불필요한 부분을 제거하고 필요한 부분만을 간결하게 만드는 것

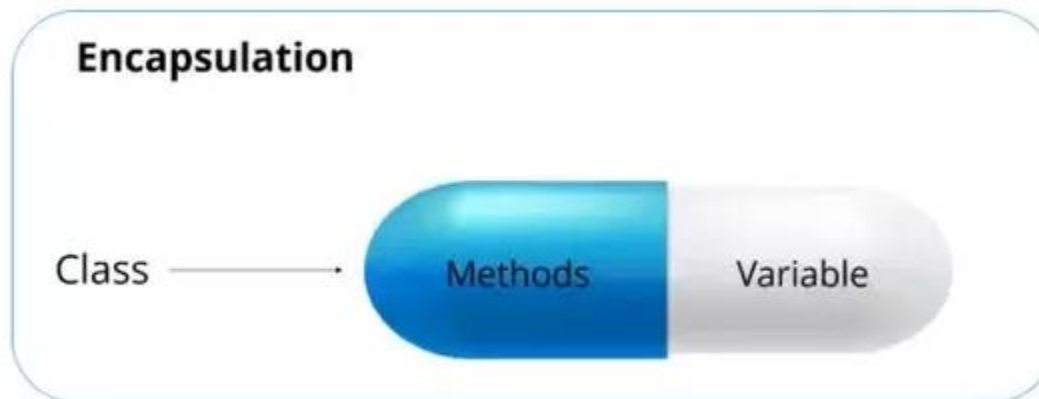


```
public class Car {  
    private int maxSpeed;  
    public String brandName;  
    public int speed;  
    public static int prodNum;  
  
    //메소드 오버로딩  
    public int speedUp() {  
        return speed += 5;  
    }  
    public int speedUp(int speed) {  
        this.speed += speed;  
        return this.speed;  
    }  
    public int speedDown() {  
        return speed -= 5;  
    }  
    public int speedDown(int speed) {  
        return this.speed -= speed;  
    }  
  
    //setter와 getter  
    public int getMaxSpeed() {  
        return maxSpeed;  
    }  
    public void setMaxSpeed(int maxSpeed) {  
        this.maxSpeed = maxSpeed;  
    }  
}
```

1. 객체지향 프로그래밍의 이해

- 캡슐화(encapsulation)

- 객체의 속성과 기능을 하나의 클래스로 묶고, 내부 구현 내용을 외부에 감추는 것

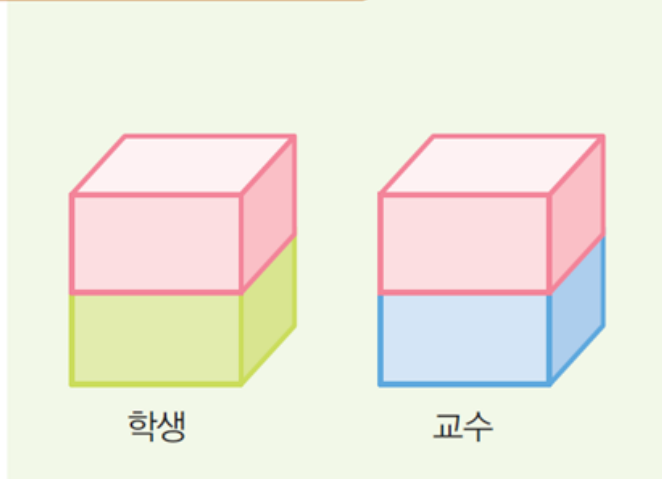


1. 객체지향 프로그래밍의 이해

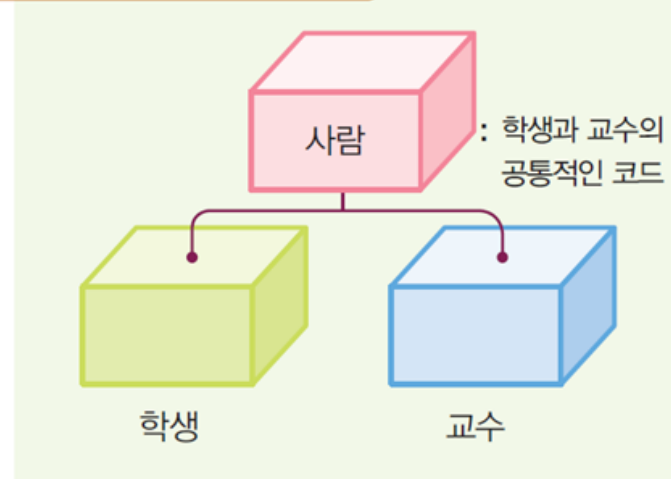
- 상속(inheritance)

- 클래스의 필드와 메소드를 다른 클래스를 만들 때 재사용하는 것
- 장점
 - 클래스의 계층적 체계화 가능
 - 확장된 클래스 생성 시 코드의 중복을 줄임

일반 프로그래밍



상속을 활용한 프로그래밍



1. 객체지향 프로그래밍의 이해

- 다형성(polymorphism)

- 한 가지 형태를 가지고 있는 클래스나 메소드가 다양한 역할을 수행하는 것

- 오버로딩(overloading)
 - 오버라이딩(overriding)



05 객체지향과 클래스

1. 객체지향 프로그래밍의 이해

2. 클래스와 객체의 생성

3. 생성자

4. static과 final

2. 클래스와 객체의 생성

- 필드 구현 및 참조
 - 필드 구현 방법

[지정자] 자료형 필드명

- 예) CreditCard 클래스의 필드 구현

```
public class CreditCard {  
    public long number;  
    public String owner;  
    ...  
}
```

필드의 자료형	기본 값
byte short int long	0
char	'\0'
float double	0.0f, 0.0
boolean	false
참조형	null

필드 기본값

2. 클래스와 객체의 생성

❖ 접근 제어자(access modifiers)

- 필드나 메소드를 참조할 수 있는 클래스 범위
- 허용 범위
 - private
 - 해당 클래스
 - default (명시하지 않은 경우)
 - 해당 클래스 + 동일 패키지 클래스
 - protected
 - 해당 클래스 + 동일 패키지 클래스 + 상속 클래스
 - public
 - 모든 클래스

2. 클래스와 객체의 생성

- 필드 구현 및 참조
 - 필드 참조 방법

변수명 . 필드명

변수는 클래스형 참조변수!

❖ 변수 선언과 동시에 객체 저장 방법

```
클래스명 변수명 = new 생성자명();
```

변수에 저장되는 것은 객체의 주소!

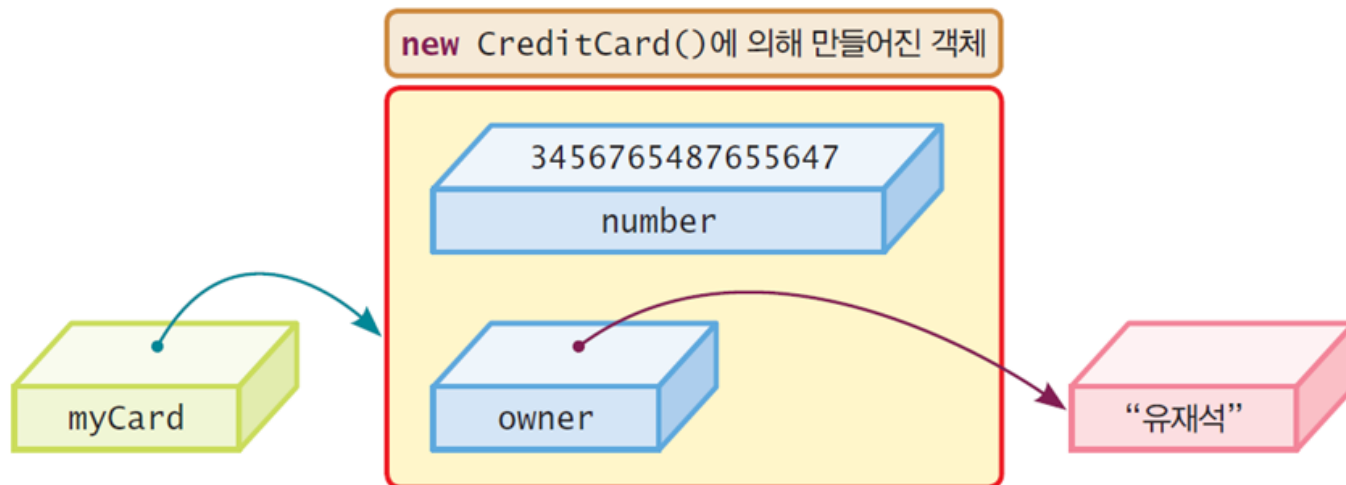
2. 클래스와 객체의 생성

- 필드 구현 및 참조

– 예) myCard 객체 생성 및 필드 참조

```
CreditCard myCard = new CreditCard();  
myCard.number = 3456_7654_8765_5647L;  
myCard.owner = "유 재석";
```

위 코드에 의해 생성된 저장 공간의 개수는? 4개



2. 클래스와 객체의 생성

[예제 5-1] 객체 생성과 필드 참조

```
public class CreditCard {  
    private long number;  
    public String owner;  
  
    public static void main(String[] args) {  
        CreditCard myCard = new CreditCard();  
  
        myCard.number = 3456_7654_8765_5647L;  
        myCard.owner = "유재석";  
        System.out.print("카드번호: " + myCard.number);  
        System.out.println(", 카드소유자: " + myCard.owner);  
    }  
}
```

2. 클래스와 객체의 생성

- 메소드 구현 및 호출
 - 메소드 구현 방법

```
[지정자] 반환자료형 메소드명 ( 매개변수 ) {  
    ...  
}
```

- 예) use, payBill, addPoint 메소드 구현

```
public void use(int amount) {  
    balance += amount;  
}  
public void payBill(int amount) {  
    balance -= amount;  
    addPoint(amount);  
}  
private void addPoint(int amount) {  
    point += amount/1000;  
}
```

구현한 메소드 addPoint()를 호출

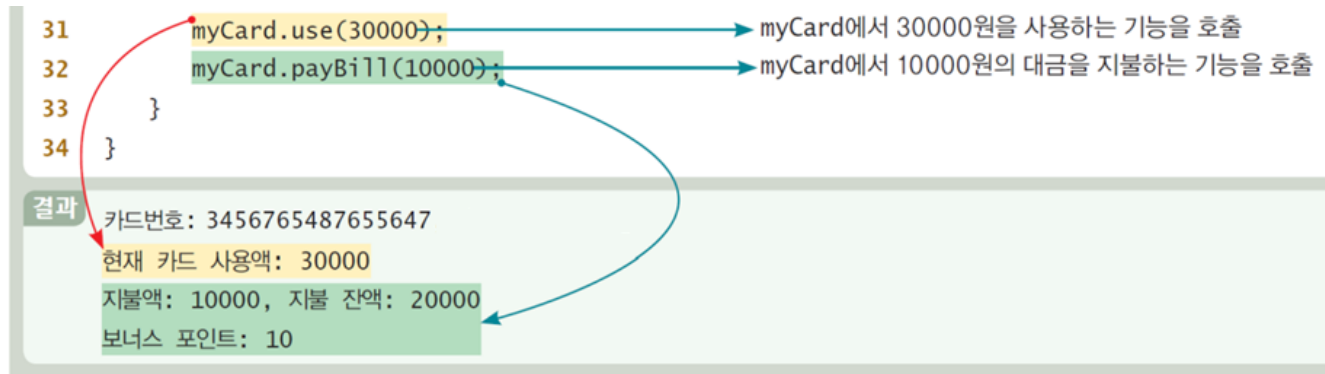
1000원에 1포인트를 추가하기 위한 연산식

2. 클래스와 객체의 생성

- 메소드 구현 및 호출
 - 메소드 호출 방법

변수명 . 메소드명 (매개변수) ;

- 예) myCard 객체의 use, payBill 메소드 호출



2. 클래스와 객체의 생성

- 객체 생성

- `new` 키워드 뒤에 생성자(constructor) 호출

```
new 생성자명()
```

생성자 이름은
클래스 이름과 같음!

- 생성된 객체 저장 방법 2가지

```
클래스명 변수명;
```

```
변수명 = new 생성자명();
```

```
클래스명 변수명 = new 생성자명();
```

2. 클래스와 객체의 생성

- Getter와 Setter

- 필드 값을 저장하고 반환하는 메소드
- 주로 접근 지정자가 private인 필드에 대해 사용자가 정의
 - getter와 setter의 접근 지정자는 주로 public
- private 필드에 대한 외부 클래스의 접근 방법 제공
- 필드 값에 대한 검증(verification) 수행

- ❖ 이클립스에서 getter와 setter 자동 구현

- Source > Generate Getters and Setters...

2. 클래스와 객체의 생성

- getter와 setter

– 예) CreditCard 클래스에서 getter와 setter 구현

```
public class CreditCard {  
    private long number; //16자리 카드번호  
    ...  
    //setter & getter  
    public long getNumber() {  
        return number;  
    }  
    public void setNumber(long number) {
```

```
        if (number < 1000_0000_0000_0000L) {  
            System.err.println("잘못된 카드 번호입니다.");  
            return;  
        }
```

```
        this.number = number;  
    }  
}
```

인자인 number 값에 대한 검증을 위한 코드이다.

인자인 number이다.

인자인 number 값을 필드 number인 this.number에 대입하는 문장

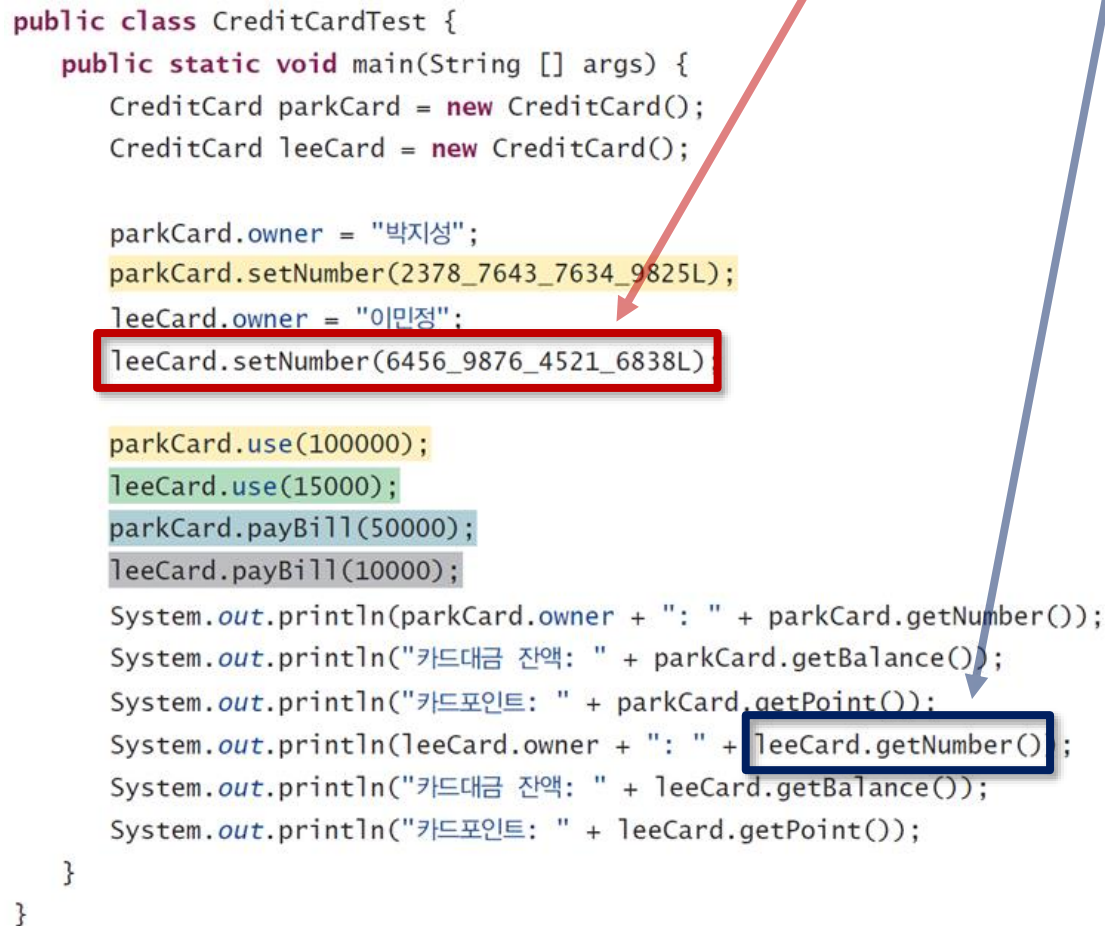
*this*는
객체 자신을 의미하는 키워드

2. 클래스와 객체의 생성

- Getter와 Setter

– 예) 외부 클래스에서 CreditCard의 getter와 setter 호출

```
public class CreditCardTest {  
    public static void main(String [] args) {  
        CreditCard parkCard = new CreditCard();  
        CreditCard leeCard = new CreditCard();  
  
        parkCard.owner = "박지성";  
        parkCard.setNumber(2378_7643_7634_9825L);  
        leeCard.owner = "이민정";  
        leeCard.setNumber(6456_9876_4521_6838L);  
  
        parkCard.use(100000);  
        leeCard.use(15000);  
        parkCard.payBill(50000);  
        leeCard.payBill(10000);  
  
        System.out.println(parkCard.owner + ": " + parkCard.getNumber());  
        System.out.println("카드대금 잔액: " + parkCard.getBalance());  
        System.out.println("카드포인트: " + parkCard.getPoint());  
        System.out.println(leeCard.owner + ": " + leeCard.getNumber());  
        System.out.println("카드대금 잔액: " + leeCard.getBalance());  
        System.out.println("카드포인트: " + leeCard.getPoint());  
    }  
}
```



05 객체지향과 클래스

1. 객체지향 프로그래밍의 이해
2. 클래스와 객체의 생성
3. 생성자
4. static과 final

3. 생성자

- 생성자(constructor)

- 객체가 생성될 때 필요한 작업을 수행하는 특별한 메소드
 - 주로 필드에 초기 값 저장 시 사용
- 반환형(void, int, long, ...) 지정하지 않음
- 이름은 클래스 이름과 동일하게 지정
- 객체 생성 시 반드시 new 키워드 다음에 생성자명 지정

```
클래스명 변수명 = new 생성자명();
```

- ❖ 이클립스에서 생성자 자동 구현

- Source > Generate Constructor using Fields...

3. 생성자

- 생성자(constructor)

– 예) Student 클래스의 생성자 정의와 생성자 호출

```
public class Student {  
    public String name;  
  
    public Student(String name) {  
        this.name = name;  
    }  
}
```

```
Student i = new Student("김민정");
```

생성자 호출 시 생성자의 매개변수와 같은 자료형으로 호출!

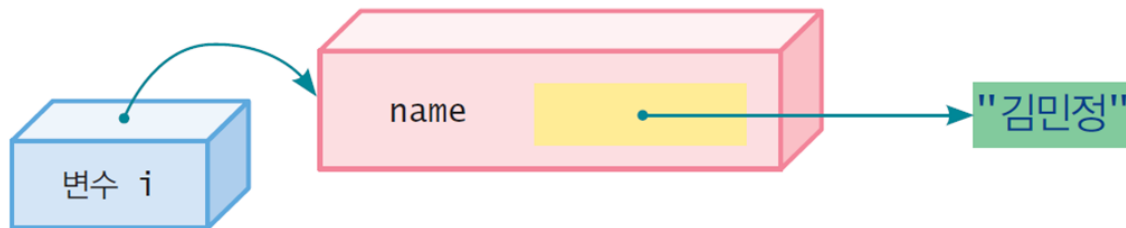
3. 생성자

- 생성자(constructor)

– 예) Student 클래스의 생성자 정의와 생성자 호출

```
Student i = new Student("김민정");
```

위 코드에 의해 생성된 저장 공간의 개수는? 3개



<메모리 구조>

3. 생성자

- 기본 생성자(default constructor)
 - 매개변수가 없는 생성자
 - 생성자를 따로 구현하지 않은 클래스는 자동으로 기본 생성자 제공

```
public class Student {  
    public String name;  
}
```

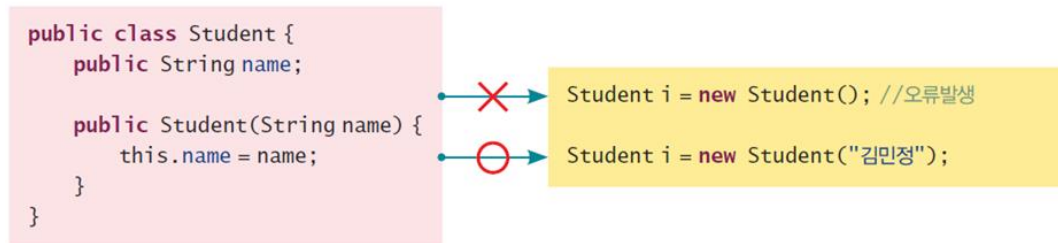


```
Student i = new Student();
```

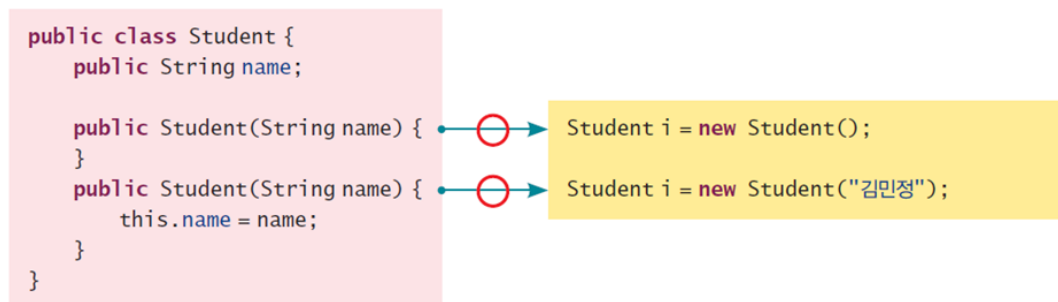
3. 생성자

- 기본 생성자(default constructor)

- 생성자가 구현되어 있을 경우, 기본 생성자 사용 못함



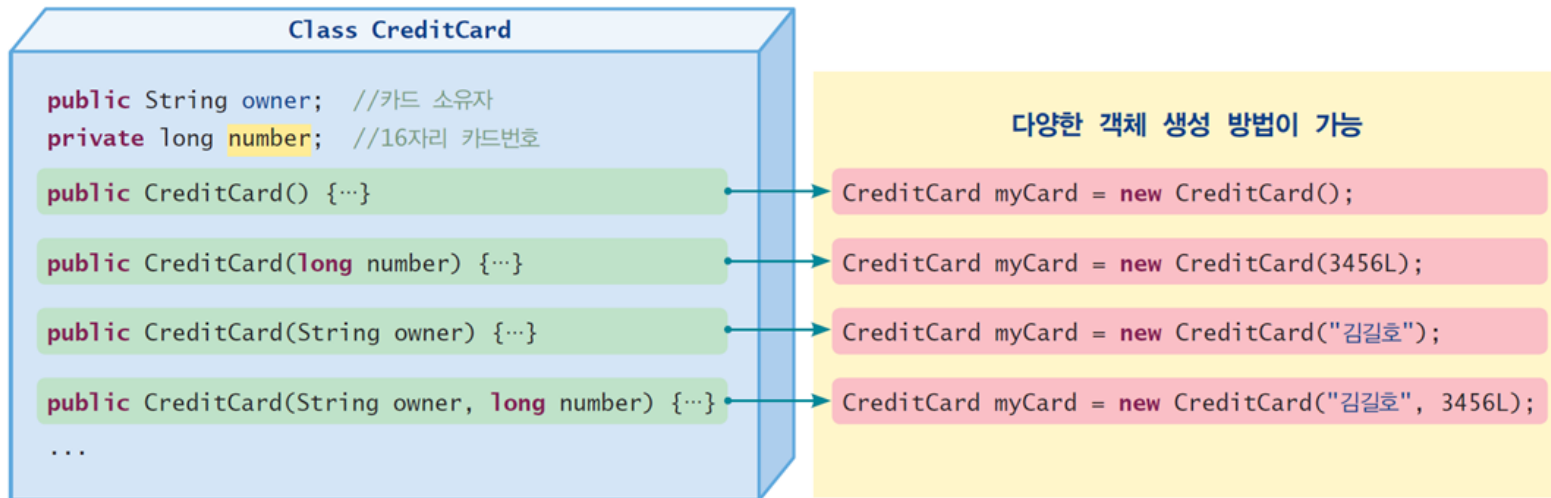
- 생성자가 구현되어 있을 경우, 기본 생성자를 사용하려면 기본 생성자 직접 구현



3. 생성자

- 생성자 오버로딩(overloading)

- 한 클래스에서 매개변수가 다른 생성자를 여러 개 만드는 것
- 객체를 생성하는 다양한 방법 제공



<생성자 오버로딩과 다양한 객체 생성 방법>

05 객체지향과 클래스

1. 객체지향 프로그래밍의 이해
2. 클래스와 객체의 생성
3. 생성자
4. Static과 Final

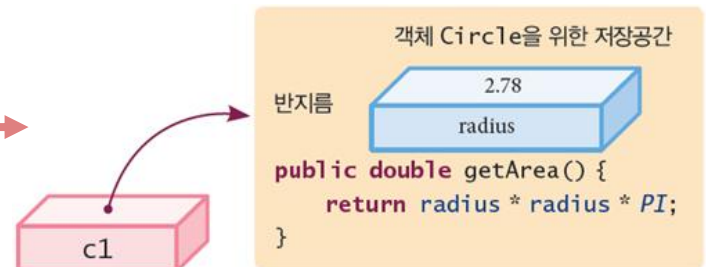
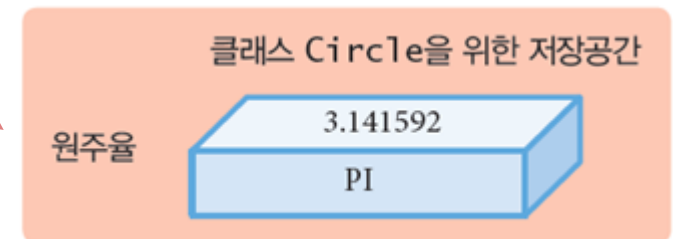
4. Static과 Final

- Static

- 필드나 메소드를 클래스 저장 공간에 하나만 생성하고, 객체에는 생성하지 않는 키워드

```
public class Circle {  
    public double radius;  
    public static double PI = 3.141592;  
  
    public Circle(double radius) {  
        this.radius = radius;  
    }  
  
    public double getArea() {  
        return radius * radius * PI;  
    }  
  
    public static void main(String[] args) {  
        Circle c1 = new Circle(2.78);  
        System.out.println(c1.getArea());  
    }  
}
```

Static 필드/메소드를
클래스 필드/메소드라 함



4. Static과 Final

- Static

- Static 필드 참조

원칙

클래스명 . 필드명

OR

허용

객체명 . 필드명

```
public class Circle {  
    public double radius;  
    public static double PI = 3.14;  
  
    public Circle(double radius) { this.radius = radius; }  
    public double getArea() { return radius * radius * PI; }  
  
    public static void main(String[] args) {  
        System.out.println("원주율: " + Circle.PI);  
  
        Circle c1 = new Circle(7.2);  
        c1.PI = 3.141592;  
  
        System.out.println("원주율: " + c1.PI);  
        System.out.println("원넓이: " + c1.getArea());  
    }  
}
```

필드 참조 시
객체명 어디에?

동일 클래스 내에서
필드 참조 시에는
객체명 또는 **this**
생략 가능!

그러나...
static 메소드에서는
생략 불가능!

4. Static과 Final

- Final

- 필드(멤버 변수)나 지역 변수 선언 시 저장 값을 더 이상 수정할 수 없도록 하는 키워드

```
public class Const {  
    public static void main(String[] args) {  
        final int maxSize = 5;  
  
        int data[] = new int[maxSize];  
        System.out.println("Array Size: " + data.length);  
        System.out.println("Array Size: " + maxSize);  
    }  
}
```

Exercise

[5-1] 다음 프로그램의 출력 결과는?

```
public class Ex {
    static int objCnt;
    int cnt = 6;

    public static void main(String[] args) {
        Ex r1 = new Ex();
        r1.objCnt = 5;
        r1.cnt = 7;

        Ex r2 = new Ex();
        r2.objCnt = 10;
        r2.cnt = 2;

        System.out.printf("%d %d %d %n", r1.objCnt, r2.objCnt, Ex.objCnt);
        System.out.printf("%d %d %n", r1.cnt, r2.cnt);
    }
}
```

Exercise

[5-1] 정답

$10 \ 10 \ 10$ $7 \ 2$

Exercise

[5-2] 아래의 선택 정렬 알고리즘을 중첩 for문을 사용하여 작성하시오.

선택 정렬은 자료 중에서 가장 앞에 위치할 값을 선택하여 자리를 바꾸는 방식으로, n 개의 자료를 정렬할 때 $n-1, n-2, \dots, 2, 1$ 번씩 비교를 반복한다. 다음은 5개의 숫자를 오름차순으로 선택 정렬하는 과정을 나타낸 것이다. 배열을 사용하여 이 과정을 구현한 자바 코드를 작성하시오.



Exercise

[5-2] 코드

```
public class SelectionSort {
    public static void main(String[] args) {
        int[] data = { 5, 9, 7, 3, 4 };
        int i, j, min, temp;

        for (i = 0; i < data.length-1; i++) {
            min = i;
            for (j = i + 1; j < data.length; j++) {
                if (data[min] > data[j])
                    min = j;
            }
            temp = data[min];
            data[min] = data[i];
            data[i] = temp;
        }

        for (i=0; i<data.length; i++)
            System.out.println("data["+i+"] : " + data[i]);
    }
}
```