NCS - Java 004 Class와 Object에 대한 이해

1. Class 란?

1. Class 란 변수와 함수의 모음을 클래스명으로 정의한 것이다

class **클래스명** {

```
final int CHECK_FLAG = "0";
int flag_none = -1;
```

변수 - 속성
Attribute

```
public void main(String args){
        int local_number = 173;
        // 로직
}
```

함수 - 기능
Method

}

2. Hello World 로 보는 Class의 이해

1. Class 란 변수와 함수의 모음을 클래스명으로 정의한 것이지만
   변수(속성:Property)가 없을 수도 있다

```
class HelloWorld {

}
```

변수 - 없음

```
public static void main(String args){
        System.out.println("Hello World!");
}
```

함수 - 기능
Method

3. 기능이 없는 Class 도 있다

1. Class 란 변수와 함수의 모음을 클래스명으로 정의한 것이지만
   함수(기능:Method)이 없을 수도 있다. C에서는 이를 구조체라 한다

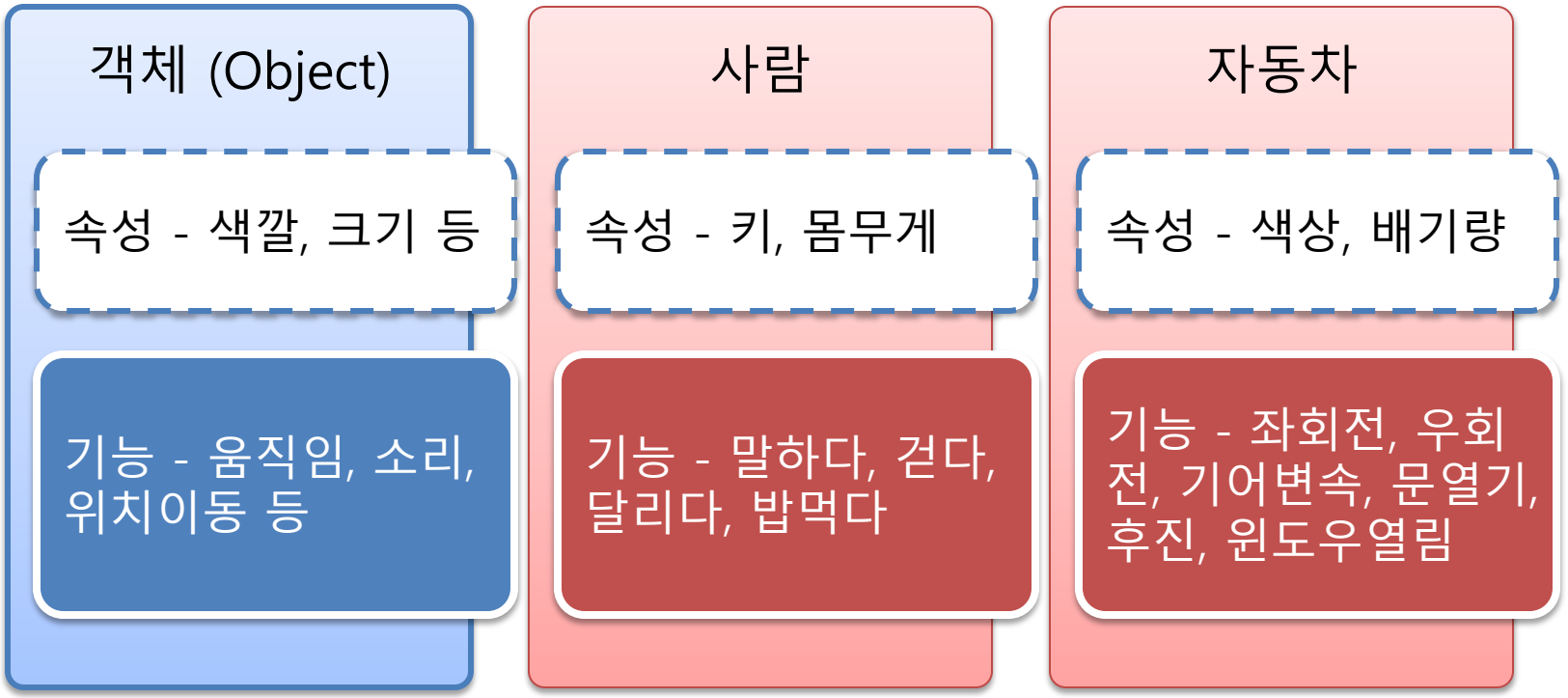
```
class Construct {

    final int CHECK_FLAG = "0";
    int flag_none = -1;
    int flag_exist = 1;
    int flag_pass = 99;

}
```

변수 - 속성
Attribute

함수 - 없음

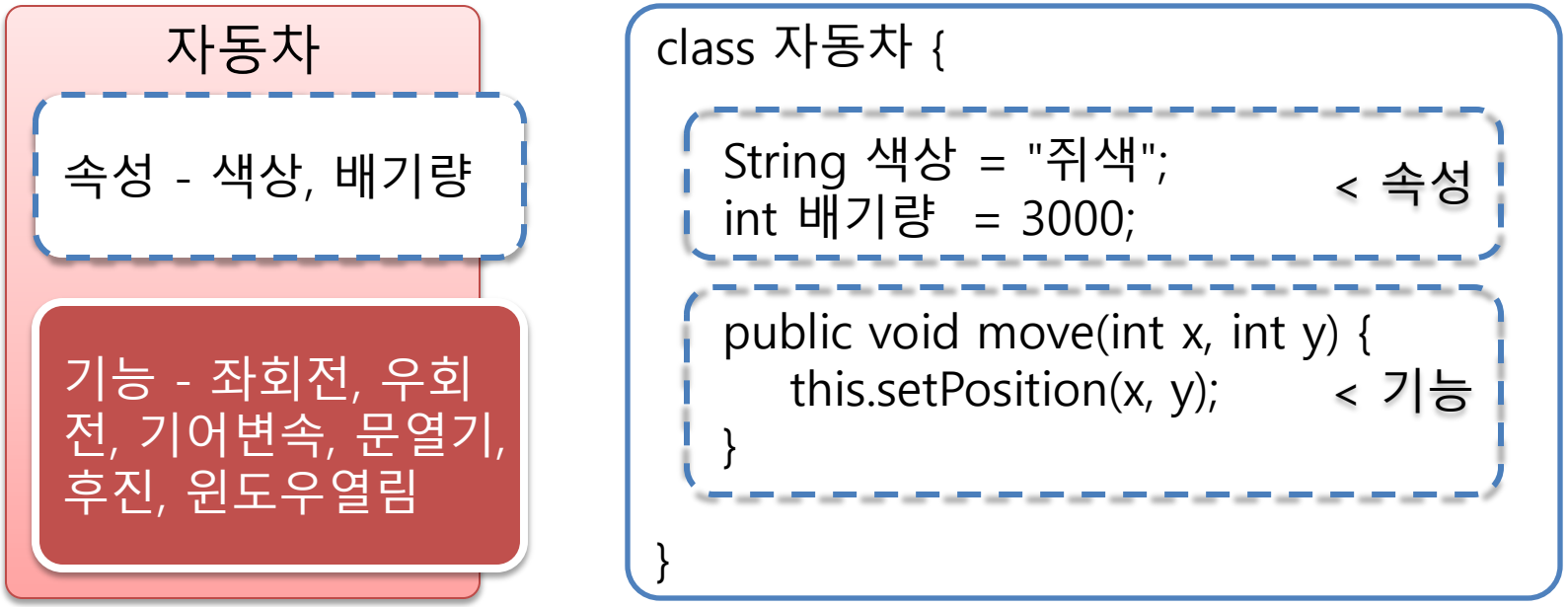
4. Object(객체) 란?

1. 프로그래밍에서 객체(Object)란 **속성**(Attribute)과 **기능**(Method)을 가지는 대상을 말한다. 속성과 동작의 개념을 이해하면 객체도 쉽게 이해할 수 있다



5. OOP 란?

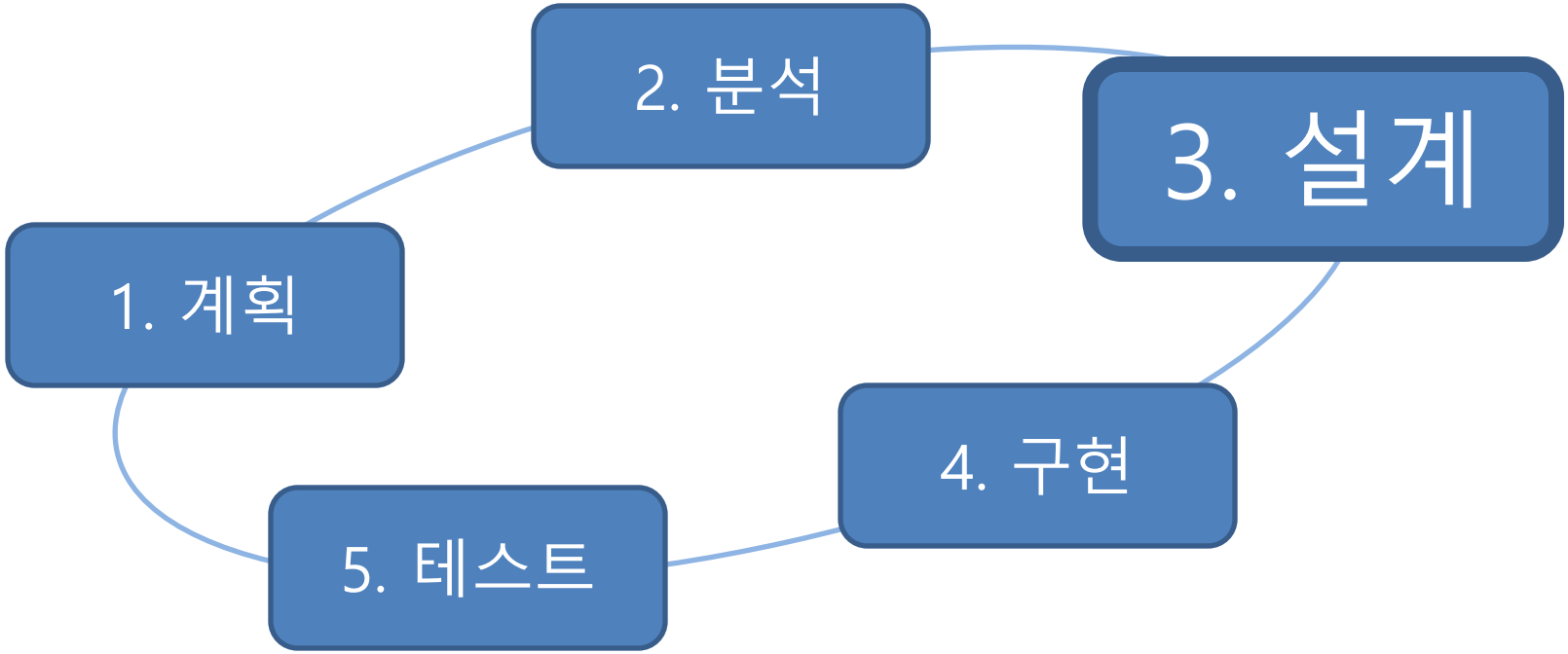
1. Object Oriented Programming 의 약자로 객체지향 프로그래밍이라 한다. 여기서 객체 지향이라 함은 앞에서 정의된 Object로 프로그램을 설계 할 때 속성과, 기능이 묶인 하나의 객체(Class) 단위로 만드는 것을 말한다.



[자동차로 보는 객체지향 프로그래밍]

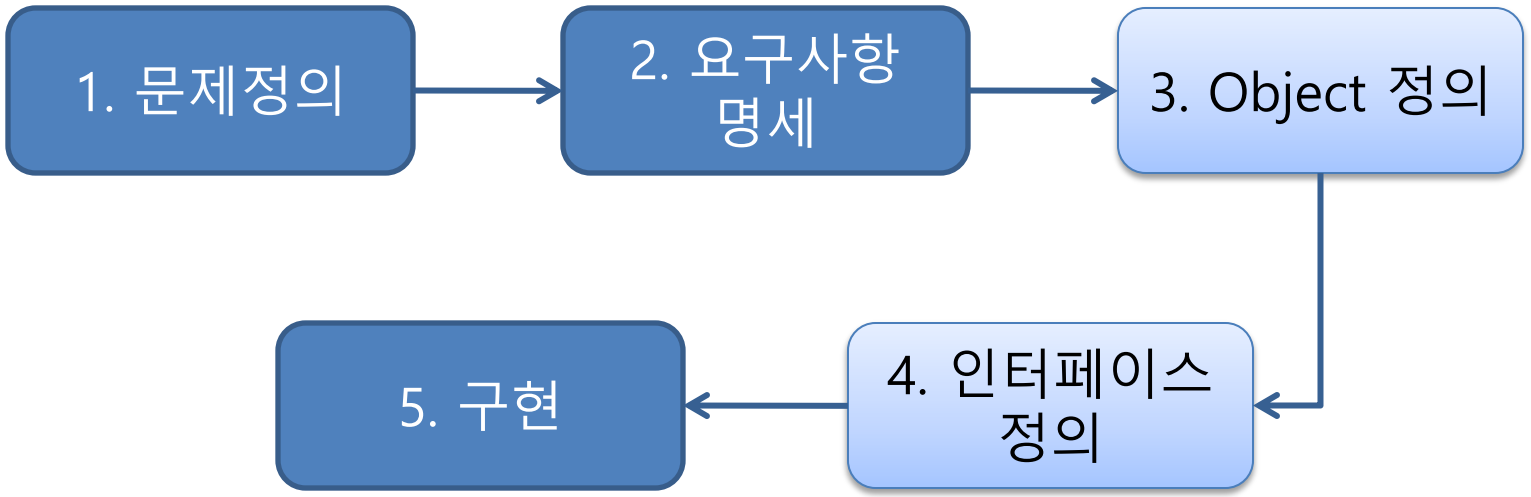
6. OOP 개발과정

1. OOP는 기본적으로 구현과 설계를 나누어서 생산성 향상과 관리의 용이성을 추구하는 데 그 목적이 있다
   OOP 개발과정을 정리하면 아래와 같은데, 그 중에서
   설계의 비중이 가장 큰 개발방식이 OOP 이다



7. OOP 설계순서

1. OOP는 설계 과정은 일반적으로 아래의 루틴을 따르는데 Object 정의 > 인터페이스 정의가 실제 프로그래밍 설계에 해당하는 작업이 된다.



8. S.O.L.I.D. - OOP 설계원칙

1. OOP 설계를 위해 지켜져야할 5가지 원칙을 각 원칙의 영문명 앞자리를 따서 SOLID 라고 한다

SRP	Single Responsibility Principle 단일 책임의 원칙
OCP	Open Closed Principle 개방-폐쇄 원칙
LSP	Liskov Substitution Principle 리스코프 교체 원칙
ISP	Interface Segregation Principle 인터페이스 격리 원칙
DIP	Dependency Inversion Principle 의존 관계 역전 원칙

8.1. SRP - 단일 책임

하나의 Class는 하나의 역할만 맡는다

높은 응집도(연관된 것끼리의 통신)와 낮은 결합도를 추구한다

1. 코드 변경의 영향이 미치는 범위가 최소화 된다
   - Class가 변경되도 다른 Class에 영향을 미치지 않는다
2. 코드 응집성이 향상된다
   - 데이터 연관성이 높은 것끼리 인터페이싱이 자주 일어난다
3. 단위테스트가 용이하다
   - 각 Class 별 테스트가 가능하다

8.2. OCP - 개방 폐쇄

확장에 대해 열려(Open) 있고, 수정에 대해 닫혀(Close) 있다

자신의 확장에는 열려있고, 주변의 변화에 대해서는 닫혀야한다

1. Class 간 통신을 위한 인터페이스가 정해지면 수정하지 않는다
   - 대신 다른 기능이 필요할 경우 확장을 통해 기능을 추가한다
   - 잘 설계된 인터페이스는 구현 후 클래스 수정을 최소화 한다
2. 표준화된 인터페이스의 경우 변경 비용이 크다
   - 해당 인터페이스를 사용한 모든 클래스가 변경되어야 한다
3. 충분히 안정화 된 이후에 적용한다
   - 모든 케이스에 대한 테스트가 완료된 후 표준으로 사용한다

8.3. LSP - 리스코프 교체

파생 Class는 상위 Class로 대체 가능해야 한다

기반 Class의 기능을 파생 Class가 포함해야 한다

1. 서브타입(파생클래스)는 언제나 자신의 기반타입(상위클래스)로 교체할 수 있어야 한다.

2. 다형성 적용
   상위클래스 a = new 하위클래스()

8.4. ISP - 인터페이스 분리

클라이언트는 자신이 쓰지 않는 인터페이스에 의존하지 않는다

특화된 여러개의 인터페이스가 범용 인터페이스 한개 보다 낫다

8.5. DIP - 의존 관계 역전

상위 모듈이 하위 모듈에 의존하면 안 된다

클라이언트는 클래스에 직접적인 의존이 아닌 추상화레이어(인터페이스)에 의존해야 한다

1. 상위모듈, 하위모듈 모두 추상화된것에 의존해야 한다
   = 추상인터페이스 상속

2. 추상화된것은 구체적인것에 의존하면 안된다

3. 자주변경되는 구체(Concrete) 클래스에 의존하면 안된다

