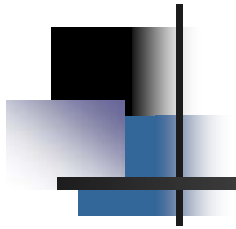


객체와 클래스–객체지향기본개념

남 광 우

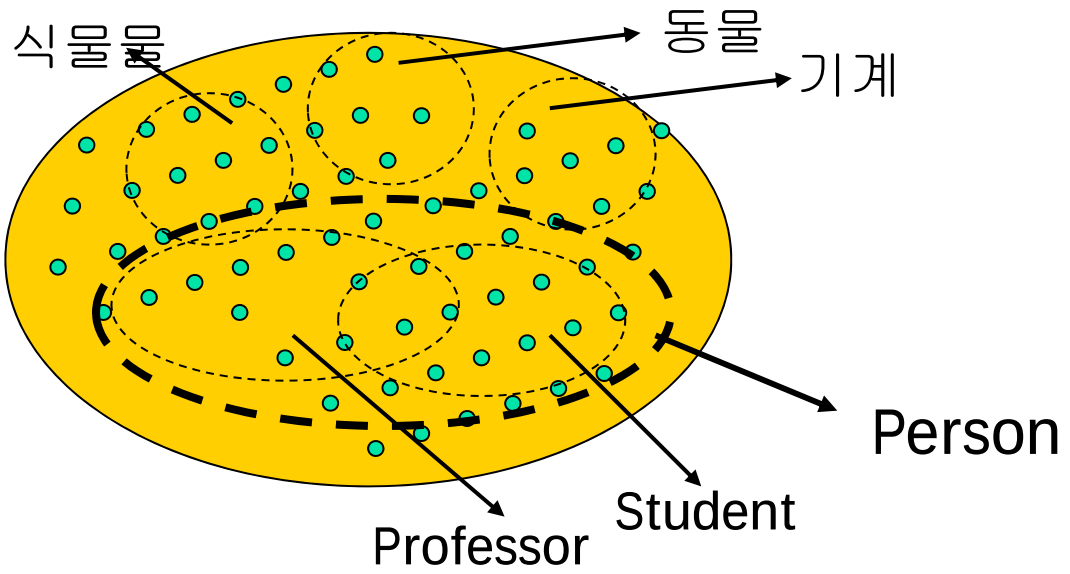
구조적 VS. 객체지향 프로그래밍

- 구조적 프로그래밍(Structured Programming)
  - 프로그램 = {함수, 함수, ..., 함수}
  - 함수 = 알고리즘 + 데이터
    - 알고리즘을 먼저 결정하고, 그 다음 조작을 쉽게 하기 위한 데이터 구조를 결정

- 객체지향 프로그래밍(Object-Oriented Prog.)
  - 프로그램 = {클래스, 클래스, ..., 클래스}
  - 클래스 = 데이터 + 함수
    - 데이터 구조를 먼저 결정하고, 이 데이터를 작동시킬 알고리즘을 결정

클래스 VS. 객체

- 객체(Object)
- 클래스(Class)



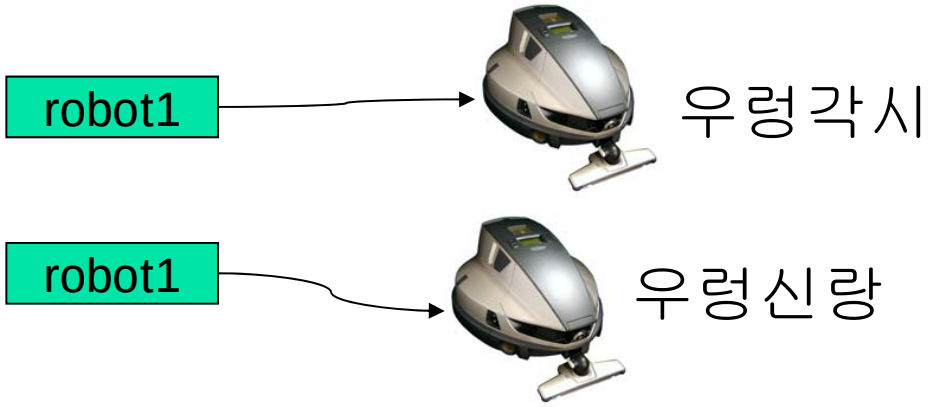
- 객체(Object)
  - 객체 = 속성(attributes) + 행위(behaviors)
  - => 변수(variables) + 메소드(methods)

- 클래스(Class)
  - 객체(object)가 실제 만들어지기 위한 템플릿
  - 동일 속성과 행위를 하는 객체들의 모음

클래스 VS. 객체

- 클래스와 객체의 관계 예

CleaningRobot robot1 = new CleaningRobot(“우렁각시”)
CleaningRobot robot2 = new CleaningRobot(“우렁신랑”)



클래스 in Java

- CleaningRobot 클래스의 예

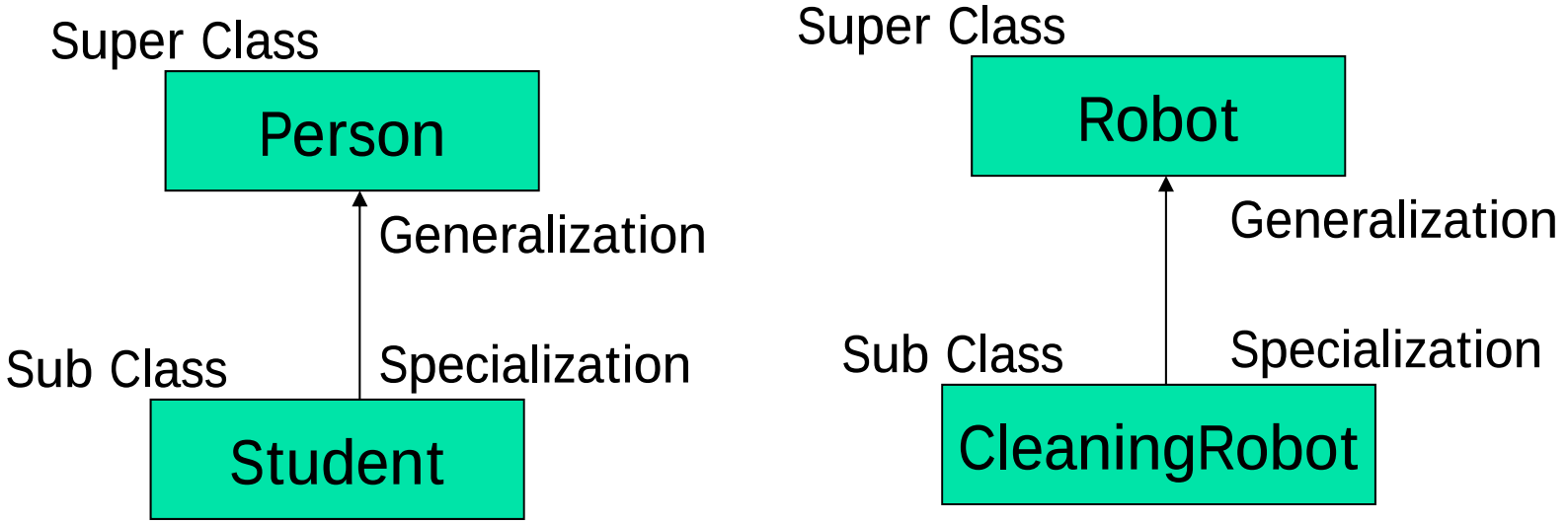
```
public class CleaningRobot
{
    private int   status;  // 0:off, 1:stop, 2:move, 3:pause
    private int   positionX;
    private int   positionY;
    private int   cleaningPower;

    public int powerOn()      {...}
    public int stop()         {...}
    public int move()         {...}
    public int moveHome()     {...}
    public int setCleaningPower()  {...}
    public int warnGarbageFull   {...}
    public static void main() {...}
}
```

메세지(Message)

- 객체의 접근 방법
  - 한 객체는 다른 객체에게 메세지를 보내 상호작용
- 메세지 교환(Message Passing)
  - 객체들 사이에 정보를 교환할 수 있는 유일한 수단
  - 한 객체가 다른 객체의 함수를 형태로 발현
- 메세지의 형태
  - 수신객체.메소드명(메소드 파라미터)
- 메세지의 예
  - robot.move()
  - robot.move(NORTH, 5);

상속(Inheritance)

- 정의
  - 기존 클래스를 새로운 클래스로 확장 하는 것
  - 신규 클래스는 기존 상위 클래스의 모든 속성과 메소드를 가짐
  - 기존 클래스의 메소드를 수정하거나 유지하며 새로운 메소드와 속성을 추가
  - 추가된 메소드는 신규 클래스에만 적용

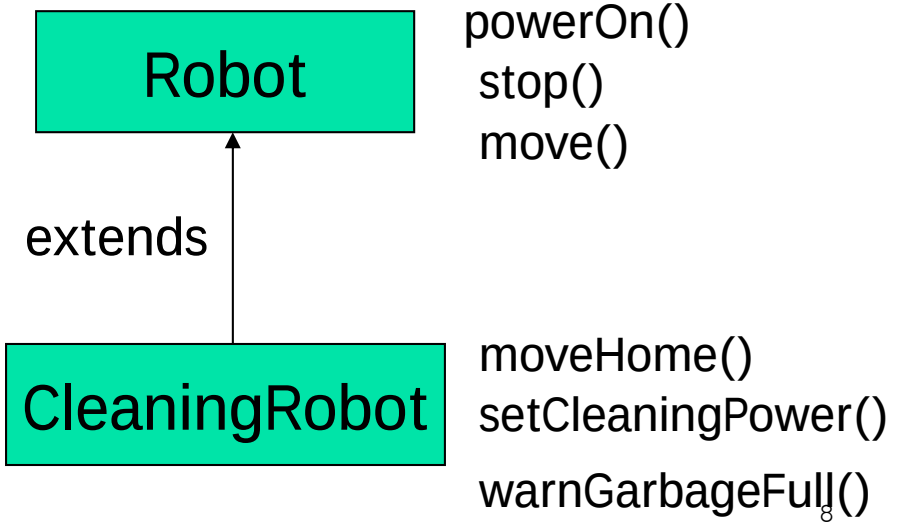


상속(Inheritance)

- 상속의 예

- Extends!

CleaningRobot extends Robot



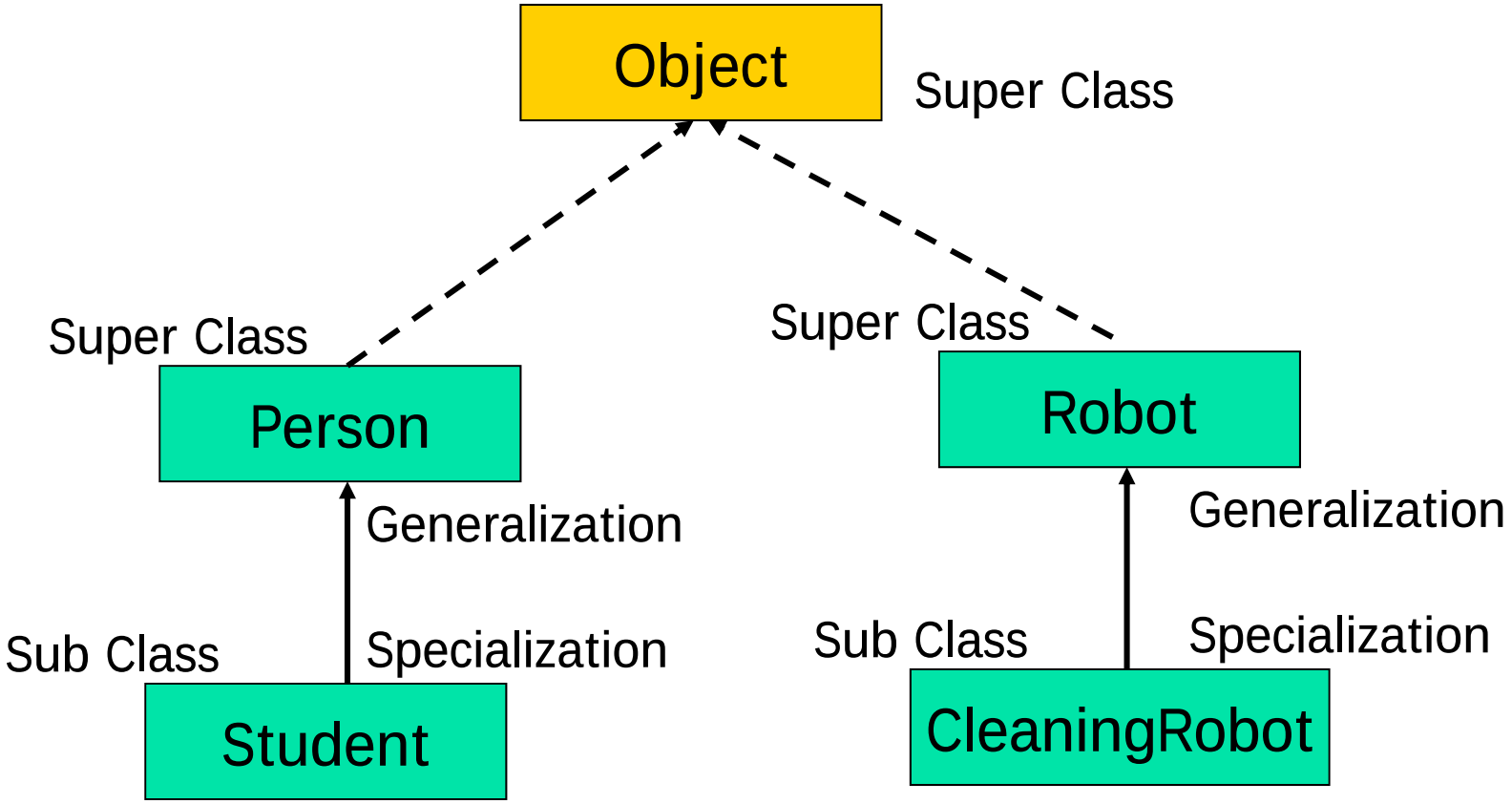
상속(Inheritance)

- 자바에서의 상속

```
public class Robot
{
        private int   status;  // 0:off, 1:stop, 2:move, 3:pause
        private int   positionX;
        private int   positionY;
        public int powerOn()      {…}
        public int stop()         {…}
        public int move()          {…}
        public static void main() {…}
}
public class CleaningRobot extends Robot
{
        private int   cleaningPower;

        public int moveHome()    {…}
        public int setCleaningPower()  {…}
        public int warnGarbageFull   {…}
        public static void main() {…}
}
```

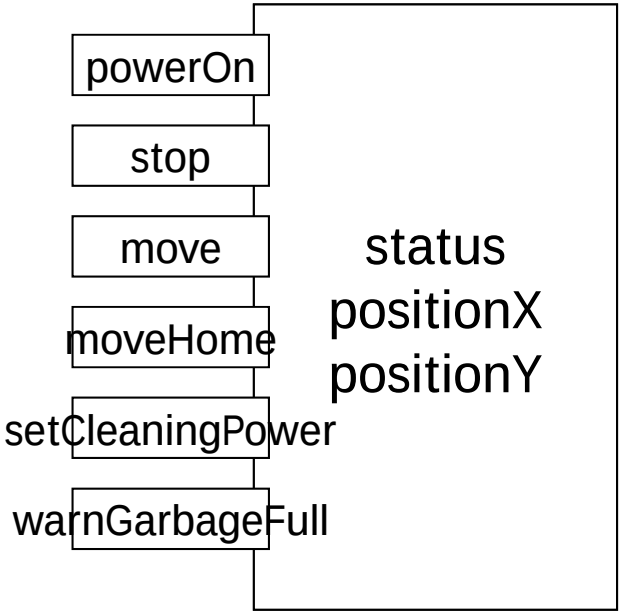
상속(Inheritance)

- 모든 클래스는 Object 클래스의 서브 클래스
  - 모든 클래스는 Object 클래스를 상속함
  - 자바의 경우는 java.lang.Object 클래스를 상속



은폐성(Encapsulation)

- 정의
  - 속성과 메소드를 하나의 패키지로 묶고, 객체의 사용자로 부터 데이터의 내부 구현을 숨기는 것
  - 데이터는 단지 메소드를 통해서만 상호작용할 수 있음
- 은폐의 목적
  - 내부 구현의 재 사용성과 신뢰성을 증가시킴
- 은폐의 장점 예
  - 로봇의 내부 부품의 변화
    - 외부 사용자는 프로그램변경 불필요



객체지향 프로그램의 설계

- 먼저, 클래스를 찾고, 각 클래스에 메소드를 추가!

- 주문처리 시스템의 예
  - 명사
    - item, order, shipping address, payment, account
    - lead to the classes Item,Order,Shipping,and so on
  - 동사
    - “add”, “ship”, “cancel”, “apply”
    - lead to the methods add,ship,cancel,and so on

예제

```
public class Bicycle {

    // the Bicycle class has
    // three fields
    public int cadence;
    public int gear;
    public int speed;

    // the Bicycle class has
    // one constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    // the Bicycle class has
    // four methods
    public void setCadence(int newValue) {
        cadence = newValue;
    }

    public void setGear(int newValue) {
        gear = newValue;
    }

    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }
}
```

```
public class MountainBike extends Bicycle {

    // the MountainBike subclass has
    // one field
    public int seatHeight;

    // the MountainBike subclass has
    // one constructor
    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    // the MountainBike subclass has
    // one method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```