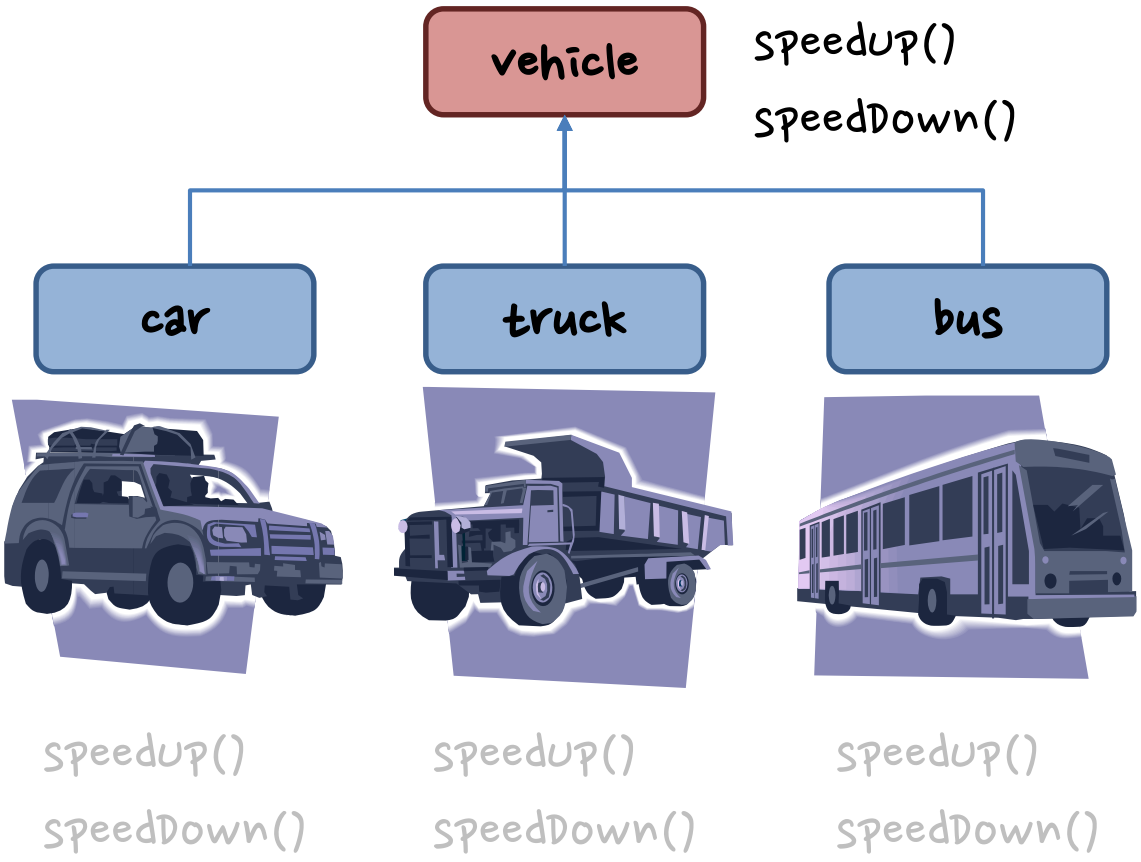
컴퓨터 프로그래밍

Inheritance

Generalization

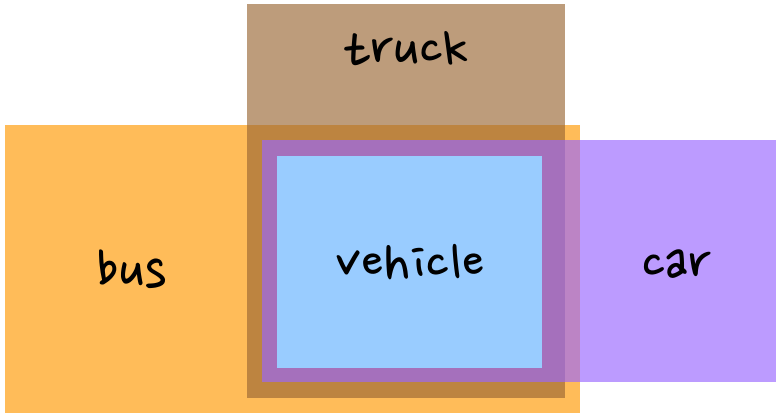


Inheritance

- 기존에 정의된 클래스에 멤버 변수/메소드를 추가하여 새로운 클래스를 정의
  - Super class (parent class, base class): 상속되는 클래스
  - Subclass (child class, derived class): 상속받는 클래스

- Syntax
  - class Subclass **extends** Superclass
    {
        … // 추가된 변수/메소드
    }

- class car extends vehicle
- class Truck extends vehicle
- class Bus extends vehicle



Example

```
class Vehicle {
    private float speed;
    public void showSpeed() {
        System.out.println("현재 속도: "+speed);
    }
}

class Truck extends Vehicle {
    private float capacity;
    public void showInfo() {
        showSpeed();
        System.out.println("적재용량: "+capacity+"t");
    }
}
```

Inheritance

- 상속 이유
  - 코드의 중복을 줄임
  - 코드의 재활용
- 상속은 is-a 관계
  - Sportscar is a car
  - Dictionary is a Book
- 포함 관계(has-a)인 경우 클래스의 멤버로 클래스를 가지도록 설계
  - Library has a Book
  - Rectangle has two Points

Instantiation

- Subclass의 생성자는 super class의 생성자를 반드시 호출

```
class Vehicle {
    private float speed;
    public Vehicle(float speed) {
        this.speed = speed;
    }
    public void showSpeed() {
        System.out.println("현재 속도: "+speed);
    }
}
```

```
class Truck extends Vehicle {
    private float capacity;
    public Truck(float speed, float capacity) {
        super(speed);
        this.capacity = capacity;
    }
    public void showInfo() {
        showSpeed();
        System.out.println("적재용량: "+capacity+"t");
    }
}
```

실습

- Vehicle class를 상속받는 Bus class를 만들고 인스턴스를 생성
  - Vehicle
    - speedUp(), speedDown() 메소드 생성
      - 0.5km/h 씩 조절
  - Bus
    - 멤버 변수: int capacity
    - 멤버 메소드: void showInfo()
      - "승차정원: "+capacity+"명"

- Vehicle class 및 Bus class의 생성자에 생성자가 호출되었음을 알리는 출력문을 넣어서 생성자가 어떻게 호출되는지 확인
- Bus class에서 Vehicle class의 멤버변수 speed에 접근할 수 있는가?

Access control

지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	o	x	x	x
default	o	o	x	x
protected	o	o	o	x
public	o	o	o	o

- private 멤버도 상속은 되나 접근은 불가
  - 함께 상속된 public(or protected) 메소드를 통해 접근

Method overriding

- Super class에 정의된 메소드를 subclass에서 재정의
- Super class의 메소드는 가려짐

```
class Vehicle {
    private float speed;
    private float weight;
    public Vehicle(float speed, float weight) {
        this.speed = speed;
        this.weight = weight;
    }
    public void showSpeed() {
        System.out.println("현재 속도: "+speed);
    }
    public float getweight() {
        return weight;
    }
}
```

```
class Truck extends Vehicle {
    private float capacity;
    public Truck(float speed, float weight,
                 float capacity) {
        super(speed, weight);
        this.capacity = capacity;
    }
    public void showInfo() {
        super.showSpeed();
        System.out.println("적재용량: "+capacity+"t");
    }
    public float getweight() {
        return super.getweight() + capacity;
    }
}
```

Assign Subclass Instance to Super class variable

- 트럭은 차량이다
  - Truck is a Vehicle

```
public static void main(String[] args) {
    Vehicle vehicle = new Truck();
    //vehicle.showInfo();   // compile error
    vehicle.showSpeed();
}
```

Method overriding & overloading

```
class AAA {
    public void rideMethod() {System.out.println("AAA's Method");}
    public void loadMethod() {System.out.println("void Method");}
}
class BBB extends AAA {
    public void rideMethod() {System.out.println("BBB's Method");}
    public void loadMethod(int num) {System.out.println("int Method");}
}
class ccc extends BBB {
    public void rideMethod() {System.out.println("ccc's Method");}
    public void loadMethod(double num) {System.out.println("double Method");}
}

public static void main(String[] args) {
    AAA ref1 = new ccc();
    BBB ref2 = new ccc();
    ccc ref3 = new ccc();

    ref1.rideMethod();
    ref2.rideMethod();
    ref3.rideMethod();

    ref3.loadMethod();
    ref3.loadMethod(1);
    ref3.loadMethod(1.2);
}
```

Result?

overriding of variables?

- 멤버 변수는 overriding 성립하지 않음
- 참조 변수의 자료형에 따라 접근 대상 결정

```
class AAA {
    public int num = 2;
}
class BBB extends AAA {
    public int num = 5;
}
class ccc extends BBB {
    public int num = 7;
}

public static void main(String[] args) {
    ccc ref1 = new ccc();
    BBB ref2 = ref1;
    AAA ref3 = ref2;

    System.out.println("ccc's ref: "+ref1.num);
    System.out.println("BBB's ref: "+ref2.num);
    System.out.println("AAA's ref: "+ref3.num);
}
```

Result?

instanceof

- 상속 관계를 바탕으로 형 변환이 가능한지 묻는 연산자
- 형 변환이 가능하면 true, 아니면 false 반환

```
class Box {
    public void simplewrap() { ··· }
}
class PaperBox extends Box {
    public void paperwrap() { ··· }
}
class GoldPaperBox extends PaperBox {
    public void goldwrap() { ··· }
}


public static void wrapBox(Box box) {
    if (box instanceof GoldPaperBox)
        ((GoldPaperBox)box).goldwrap();
    else if (box instanceof PaperBox)
        ((PaperBox)box).paperwrap();
    else
        box.simplewrap();
}
```

overriding으로 처리한다면?

Merit of Inheritance and Overriding

- 상속 관계인 일련의 클래스에 대한 공통적인 규약 정의
  - e.g. Truck, Bus 클래스에 대해
    동일한 방식(Vehicle 클래스)으로 배열에 저장 및 메소드 호출

```
public static void main(String[] args) {
    Vehicle[] vehicles = new Vehicle[2];
    vehicles[0] = new Truck(0, 10);
    vehicles[1] = new Bus(0, 10);
    for (int i=0 ; i<vehicles.length ; i++)
        vehicles[i].showSpeed();
}
```

final class / method

- 클래스에 final 키워드를 붙이면, 클래스의 상속을 허용하지 않는다
- 멤버 메소드에 final 키워드를 붙이면 해당 메소드의 overriding을 허용하지 않는다

```
final class NotInheritable {
    ...
}
```

```
class Inheritable {
    final void NotOverridable(...) { ... }
    ...
}
```

abstract class

- 객체의 생성을 허용하지 않는 클래스 (추상화 클래스)

```
abstract class Vehicle {
    private float speed;
    private float weight;
    public Vehicle(float speed, float weight) {
        this.speed = speed;
        this.weight = weight;
    }
    public void showSpeed() {
        System.out.println("현재 속도: "+speed);
    }
    public float getWeight() {
        return weight;
    }
    public abstract void showInfo();
}
```

```
class Truck extends Vehicle {
    private float capacity;
    public Truck(float speed, float weight,
                 float capacity) {
        super(speed, weight);
        this.capacity = capacity;
    }
    public void showInfo() {
        super.showSpeed();
        System.out.println("적재용량: "+capacity+"t");
    }
    public float getWeight() {
        return super.getWeight() + capacity;
    }
}
```

abstract class

- Vehicle 클래스는 인스턴스화할 의도가 없다
  - Vehicle 클래스가 인스턴스화 된다면 실수이며, 이것을 막고 싶다
- showInfo method?
  - overriding 관계를 위한 method
  - abstract로 선언 가능
- abstract method를 포함하는 클래스는 abstract로 선언
- 인스턴스 생성은 불가능, 참조 변수 선언은 가능
  - Vehicle v = new Vehicle(…); // 불가
  - Vehicle v = new Truck(…); // 가능
- abstract 클래스를 상속하는 subclass는 반드시 abstract method를 overriding 해야 한다
  - Truck / Bus 클래스의 showInfo method를 반드시 구현

Interface

- 완벽한 abstract class
  - 모든 method가 abstract인 class는 interface로 정의 가능
  - interface 내 선언된 변수는 무조건 public static final
  - interface 내 선언된 메소드는 무조건 public abstract
  - interface도 참조 변수 선언 및 method overriding 가능

```
public interface MyInterface {
    public void myMethod();
}
public interface YourInterface {
    public void yourMethod();
}
```

```
public interface SuperInterf {
    public void supMethod();
}
public interface SubInterf extends SuperInterf {
    public void subMethod();
}
```

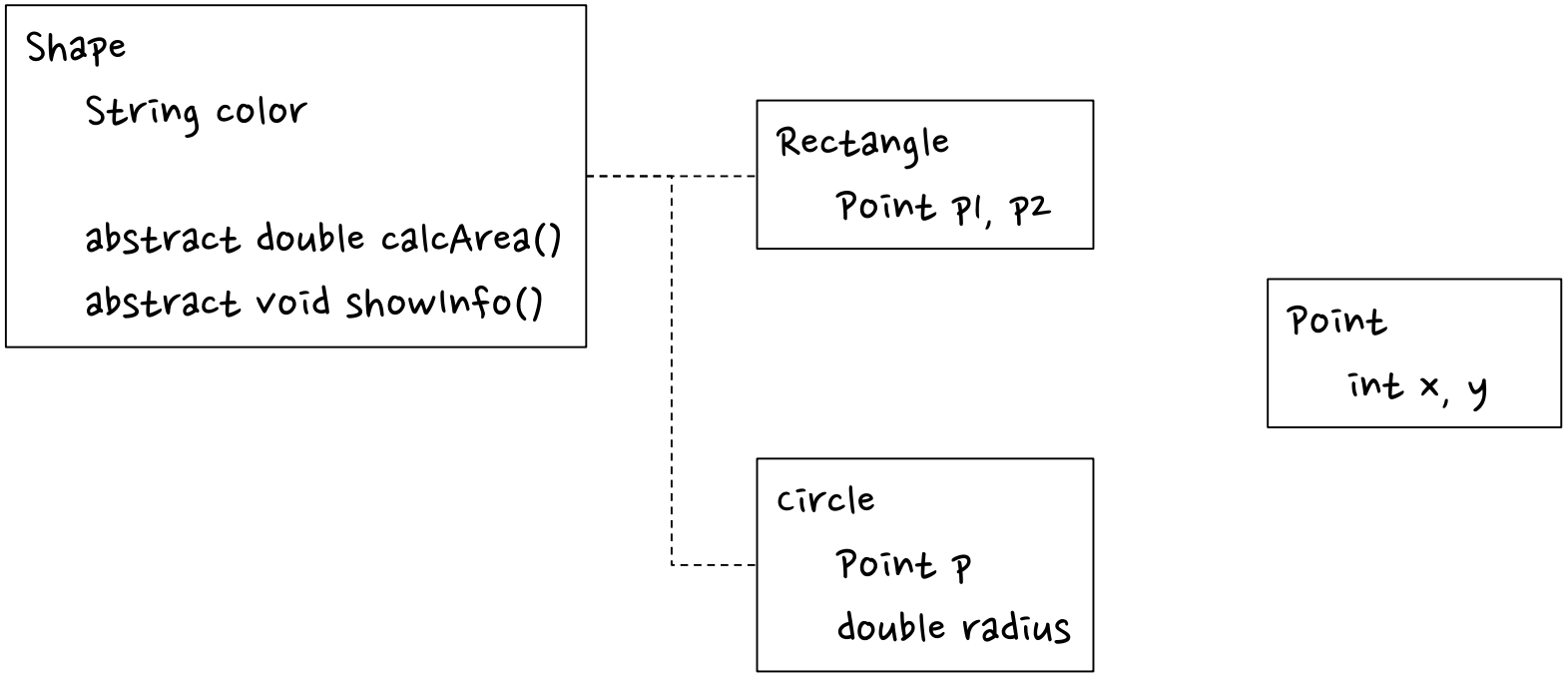
```
class ourclass implements MyInterface, YourInterface {
    public void myMethod() { ··· };
    public void yourMethod() { ··· };
}
```

Summary

- Inheritance (상속)
  - 기존에 정의된 클래스에 멤버 변수/메소드를 추가하여 새로운 클래스를 정의
  - is-a 관계의 클래스 간에 활용
- Method overriding
  - Subclass에서 super class에서 정의한 method를 재정의
- Inheritance & overriding
  - 상속 관계인 일련의 클래스에 대한 공통적인 규약 정의
- abstract class
  - 추상적 개념의 abstract class를 통해 공통 요소 도출
  - abstract class를 상속 받아 인스턴스화

실습

- 도형 관리 프로그램 구현
  - abstract class인 Shape class를 상속 받는 Rectangle과 circle class 구현



calcArea(): 도형의 넓이 계산

showInfo(): 도형의 정보 출력 (e.g. red 사각형, black 원)

실습

- 도형 관리 프로그램 구현
  - abstract class인 Shape class를 상속 받는 Rectangle과 circle class 구현

```
public static void main(String[] args) {
    Shape[] s = new Shape[3];
    s[0] = new Rectangle(···);
    s[1] = new Rectangle(···);
    s[2] = new circle(···);

    for (int i=0 ; i<s.length ; i++) {
        System.out.println(s[i].showInfo());
        System.out.println("넓이: " + s[i].calcArea());
    }
```

```
red 사각형
넓이: 10.0
blue 사각형
넓이: 30.0
black 원
넓이: 314.1592
```