상속

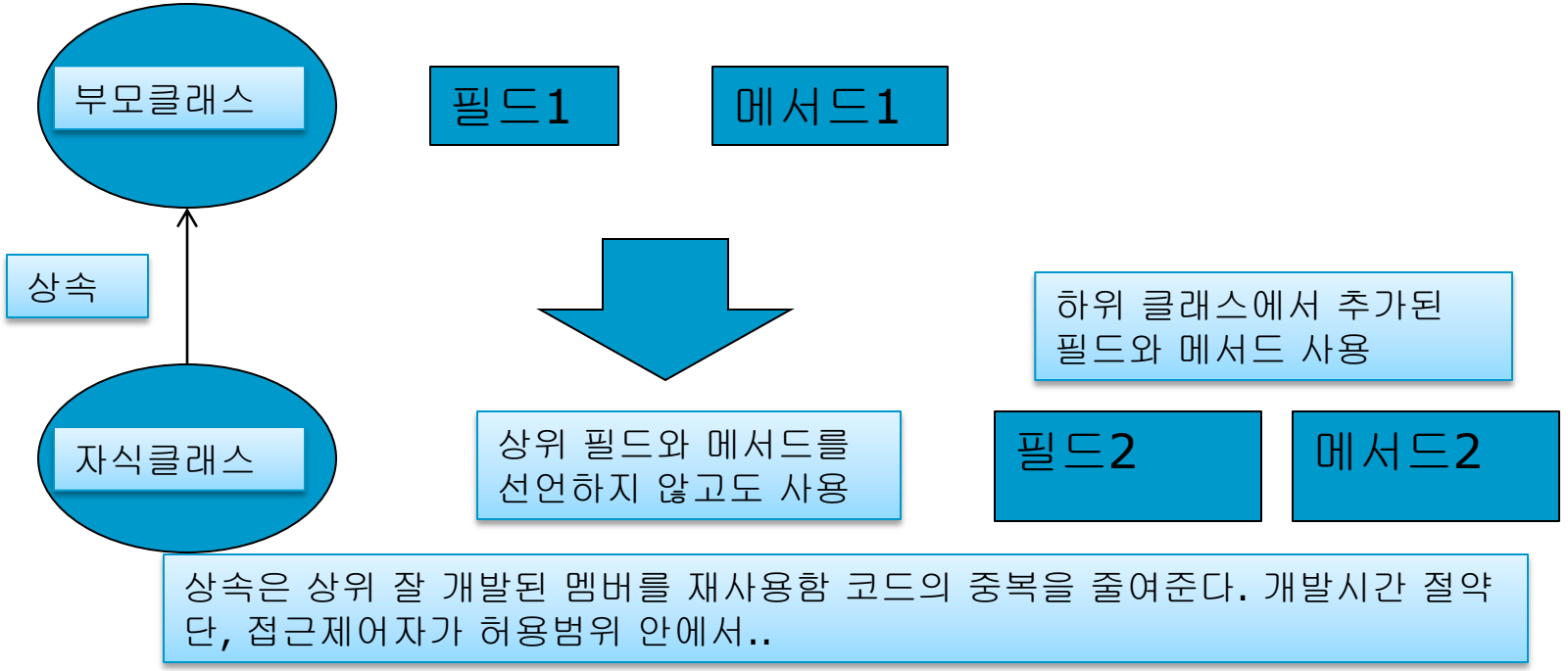
목표:

- 자바 프로그램에서 상속의 의미하는 바와 실제코드를 작성할 수 있다!
- 오버로딩(중복정의)와 오버라이딩(재정의**riding-**편성)의 차이를 알고 구분해서 구현할 수 있다.
- 다형성(**polymorphism**폴리 모피즘)의 개념을 이해하고, 활용할 수 있다.
- 추상클래스와 인터페이스 개념을 이해하고 활용할 수 있다.

생각해봅시다:

상속 이란?

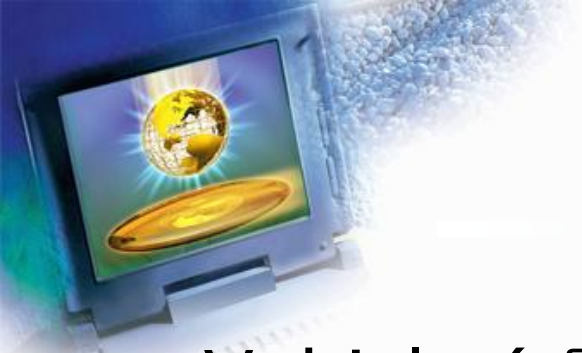
- 상속(Inheritance)은 부모가 자식에서 물려주는 행위를 현실에서도 개념을 가지고 있듯이, 자바에서도 클래스간 상속을 하면, 자식클래스는 부모클래스의 멤버를 활용할 수 있는 것을 말한다. 부모클래스를 상위 클래스(super)라고 부르고, 자식클래스를 하위 클래스, 파생 클래스라고 한다.



클래스 상속

- 부모 클래스를 선언하고, 상속받을 클래스에 extends 키워드를 이용해서 상속을 받는다.
- 단, 기본 클래스는 단일, 상속만 허용한다.

```
class 부모클래스{ 멤버....}
class 자식클래스 extends 부모클래스, 부모클래스2(X) {
    추가할 멤버...
}
```



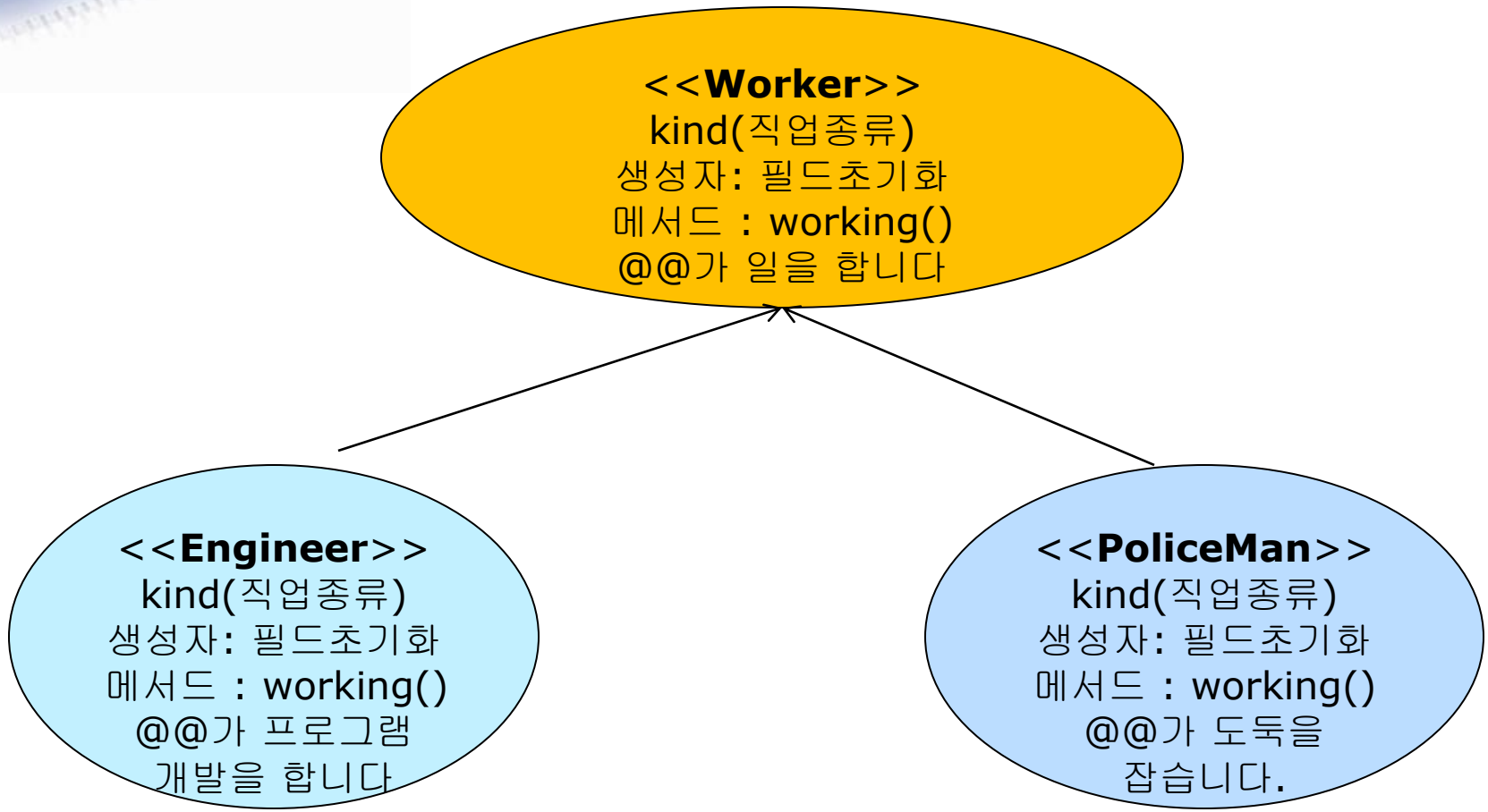
확인예제(숙제)

- Vehicle (탈것)
  - 필드: 탈것의 종류, 최고속도
  - 메서드 : showInfo()
    - @@@인데, 최고 속도가 @@@ 이다.
- Car : Vehicle을 상속 받음.
  - 필드 : 타는 사람 수
  - 메서드 : driverCar()
    - @@@인데, 최고 속도가 @@@ 이다.
    - 타는 사람은 @@@명 입니다.

method overriding

- 부모 클래스에 정의된 메서드를 그대로 사용하는 경우도 있지만, 자식클래스가 부모클래스이 메서드를 변형하여 각 자식클래스의 용도 맞게 사용하는 경우가 많다.
- 이 때, 부모 클래스의 동일 메서드를 수정하여 사용하는 것을 메서드 오버라이딩이라고 한다.
- 기본 코드 형태..
  - class 상위클래스{
    - void method01(){} // 오버라이딩할 메서드
  - class 자식클래스 extends 상위클래스{
    - void method01(){} //오버라이딩 메서드

확인예제


<<Worker>>
kind(직업종류)
생성자: 필드초기화
메서드 : working()
@@가 일을 합니다
<<Engineer>>
kind(직업종류)
생성자: 필드초기화
메서드 : working()
@@가 프로그램
개발을 합니다
<<PoliceMan>>
kind(직업종류)
생성자: 필드초기화
메서드 : working()
@@가 도둑을
잡습니다.

상속에서 final

- 클래스명 앞에 final
  - 상속할 수 없는 클래스 선언..
  - public final class 상위클래스명{}
    - 하위 클래스에서 상속을 할 수 없게 처리.
    - public class 하위클래스 extends 상위클래스(X)
- 메서드명 앞에 final
  - overriding할 수 없는 메서드 선언.
    - 하위 클래스에서 상위 클래스의 메서드를 재정의를 할 수 없게 처리.
    - public class 상위클래스{
    - public final void show(){}
    - public class 하위클래스 extends 상위클래스{
    - public void show(){} (X)

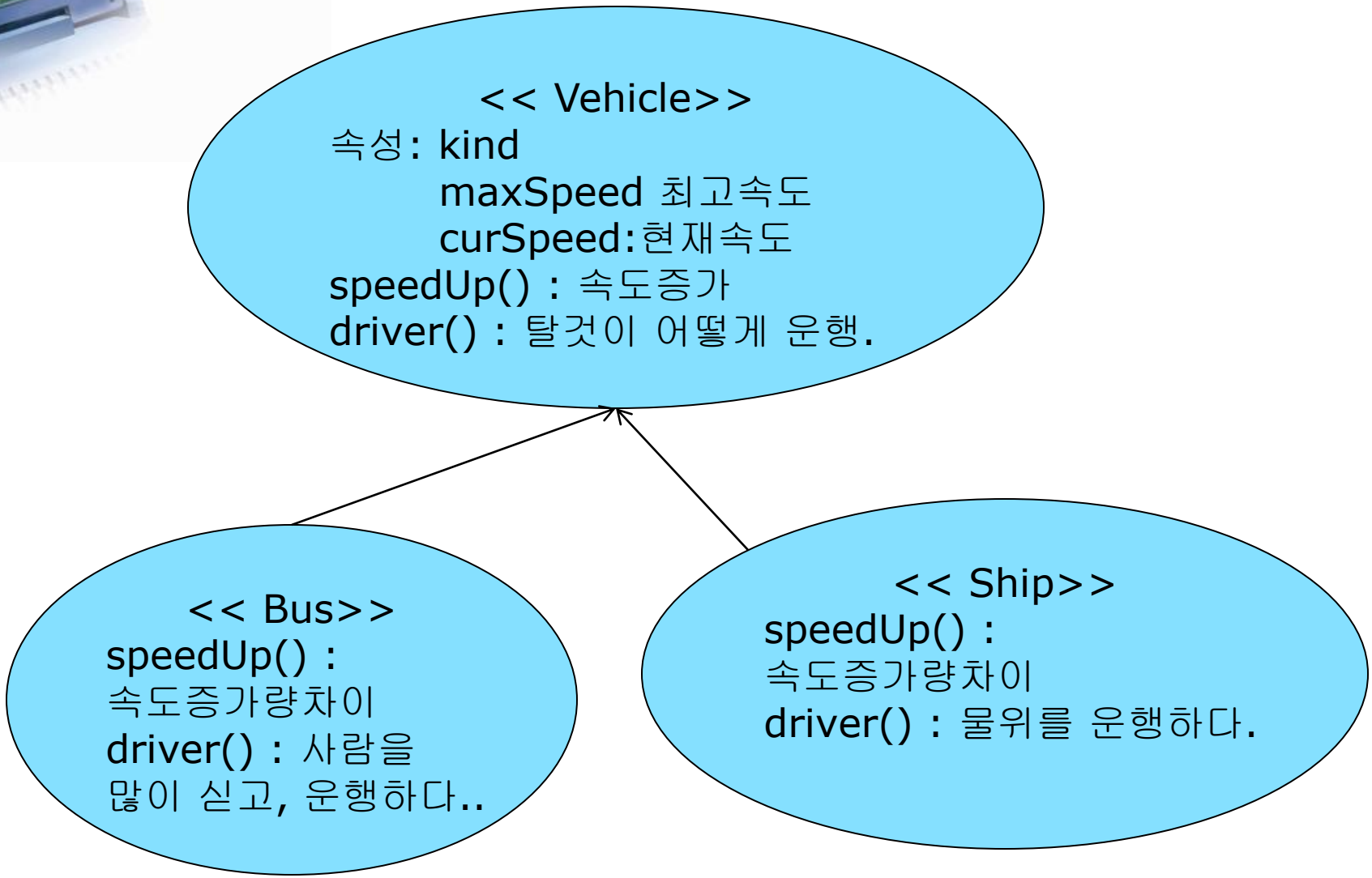
접근제어자 protected

- public > protected > [default] > private
- 전체다 부분적 전체다 패키지만 클래스
- 접근제어범위는 상속관계있는 멤버만 패키지 상관 없이 접근이 가능하다.

Polymorphism(다형성)

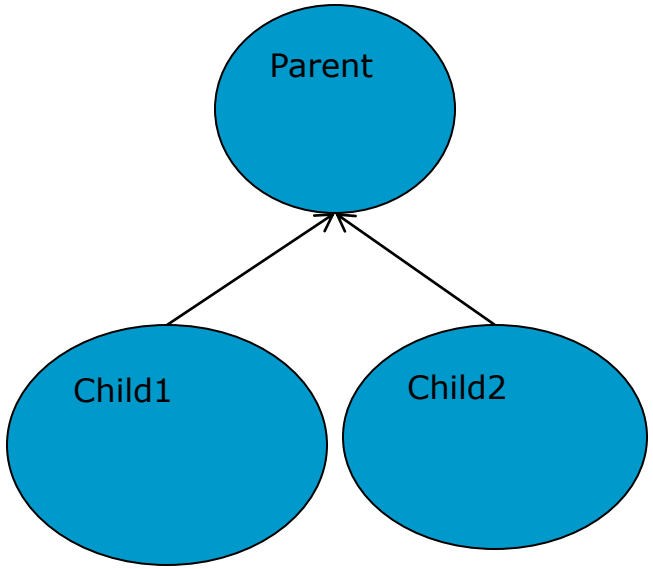
- 상속관계에서 부모타입에서 모든 자식 객체가 대입될 수 있다.
- 상속관계 클래스 정의
  - class Father{}
  - class Son extends Father{}
  - class Daughter extends Father{}
- main이나 다른 클래스에서 객체 생성시, 타입 대입 가능
  - Father f1 = new Son();
  - Father f2 = new Daughter ();
  - 부모클래스 변수 = 자식클래스타입() : 자동 타입 변환

다형성 확인예제


<< Vehicle>>
속성: kind
maxSpeed 최고속도
curSpeed:현재속도
speedUp() : 속도증가
driver() : 탈것이 어떻게 운행.
<< Bus>>
speedUp() :
속도증가량차이
driver() : 사람을
많이 싣고, 운행하다..
<< Ship>>
speedUp() :
속도증가량차이
driver() : 물위를 운행하다.

강제 타입 변환(Casting)

- 상위type 참조변수 = 하위객체(); [promote]
- 하위type 참조변수 = (하위type)상위객체();[casting]
- class Parent{
  - void getInfo();
- }
- class Child extends Parent{
- void childInfo();
- }
- main()
  - Parent p = new Child();
  - p.childInfo(); // 다형성에서 추가적인 메서드는 활용(X)
  - Child c = (Child)p;
  - c.childInfo(); // 하위객체의 특정 메서드를 활용(O)



객체 타입 확인(instanceof)

- 상속의 계층구조로 여러 가지 하위객체들을 만들어 갈 수 있다. 이 때, 특정한 객체에 해당 상위 객체의 소속을 때만, 다형성이나 casting 처리할 수 있다. 이 때, 해당 객체의 상위 클래스의 상속받은 하위 클래스를 확인 할 때, instanceof
  - boolean isInherit = 상위객체 instanceof 하위객체;
  - if(parent instanceof Child){
    - Child child = (Child)parent;
  - }

추상클래스

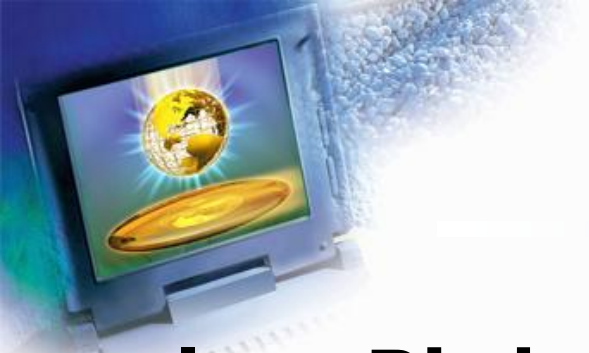
- 추상(abstract) : 실제 간의 공통되는 특성을 추출하는 것을 말한다.
  - 새, 곤충, 물고기 : 실제 ➔ 공통 특성 ➔ 동물
  - 삼성, 현대, LG ➔ 회사
- 추상클래스란 실제 클래스의 **공통**적인 부분을 추출해서 만드는 것을 말한다. 실제클래스가 아니기에 **객체 생성을 하지 못하는 것**을 말한다.
- 추상 클래스 구성 내용
  - 공통메서드 ex) eat(){ ...먹는 처리.. } ➔ 동일.
  - 오버라이딩(재정의 할 메서드 선언);
  - move(){ 다양한 형태로 하위에서 처리...}
    - abstract void move(); //{} **body부분이 없음.**
    - ➔ 추상메서드
  - 추상메서드가 하나라도 있는 클래스는 추상클래스

추상클래스

- 추상 클래스를 통한 하위 객체 생성.
  - main()
  - 추상클래스 객체변수 = new 실제객체();

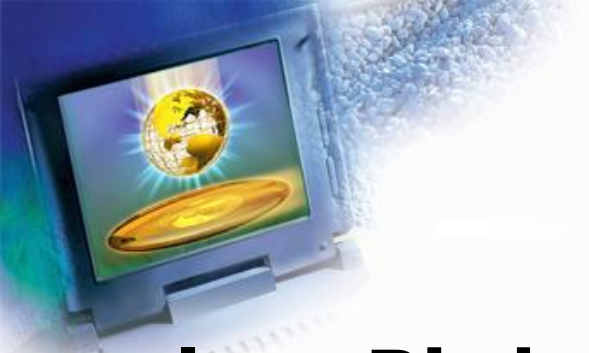
추상 클래스 기본예제

```
abstract class Animal01{
    // 공통 메서드..
    public void eat(){
    System.out.println("맛있게 먹다!!");
    }
    // 하위클래스에서 재정의할 메서드..
    // 추상메서드는 앞에 abstract modifier가 붙고,
    // 메서드의 {}<body> 부분이 없다..
    // 클래스 메서드 가운데 추상메서드가 하나라도 있으면,
    // 추상 클래스가 된다. 추상클래스는 abstract를 붙여
       준다.
    public abstract void move();
}
```



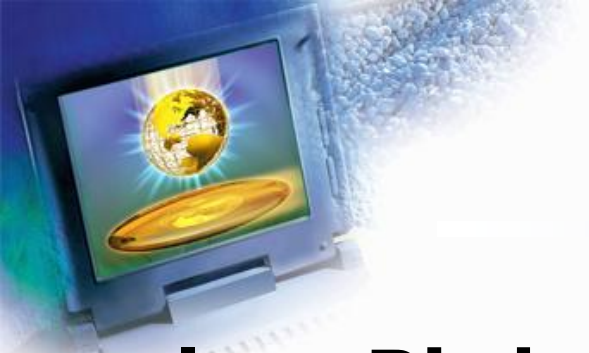
추상 클래스 기본예제

```
class Bird extends Animal01{
    // 추상클래스를 상속받은 하위 클래스는
    // 반드시 추상메서드를 재정의 하여야 한다.
    // 기능 메서명의 통일을 유지할 수 있다.
    @Override
    public void move() {
    // TODO Auto-generated method stub
    System.out.println("Flying!!! Sky~~");
    }
}
```



추상 클래스 기본예제

```
class Bird extends Animal01{
    // 추상클래스를 상속받은 하위 클래스는
    // 반드시 추상메서드를 재정의 하여야 한다.
    // 기능 메서명의 통일을 유지할 수 있다.
    @Override
    public void move() {
    // TODO Auto-generated method stub
    System.out.println("Flying!!! Sky~~");
    }
}
```



추상 클래스 기본예제

```
class Bird extends Animal01{
    // 추상클래스를 상속 받은 하위 클래스는
    // 반드시 추상메서드를  재정의 하여야  한다.
    // 기능 메서드명의  통일을 유지할 수 있다.
    @Override
    public void move() {
    // TODO Auto-generated method stub
    System.out.println("Flying!!! Sky~~");
    }
}
```

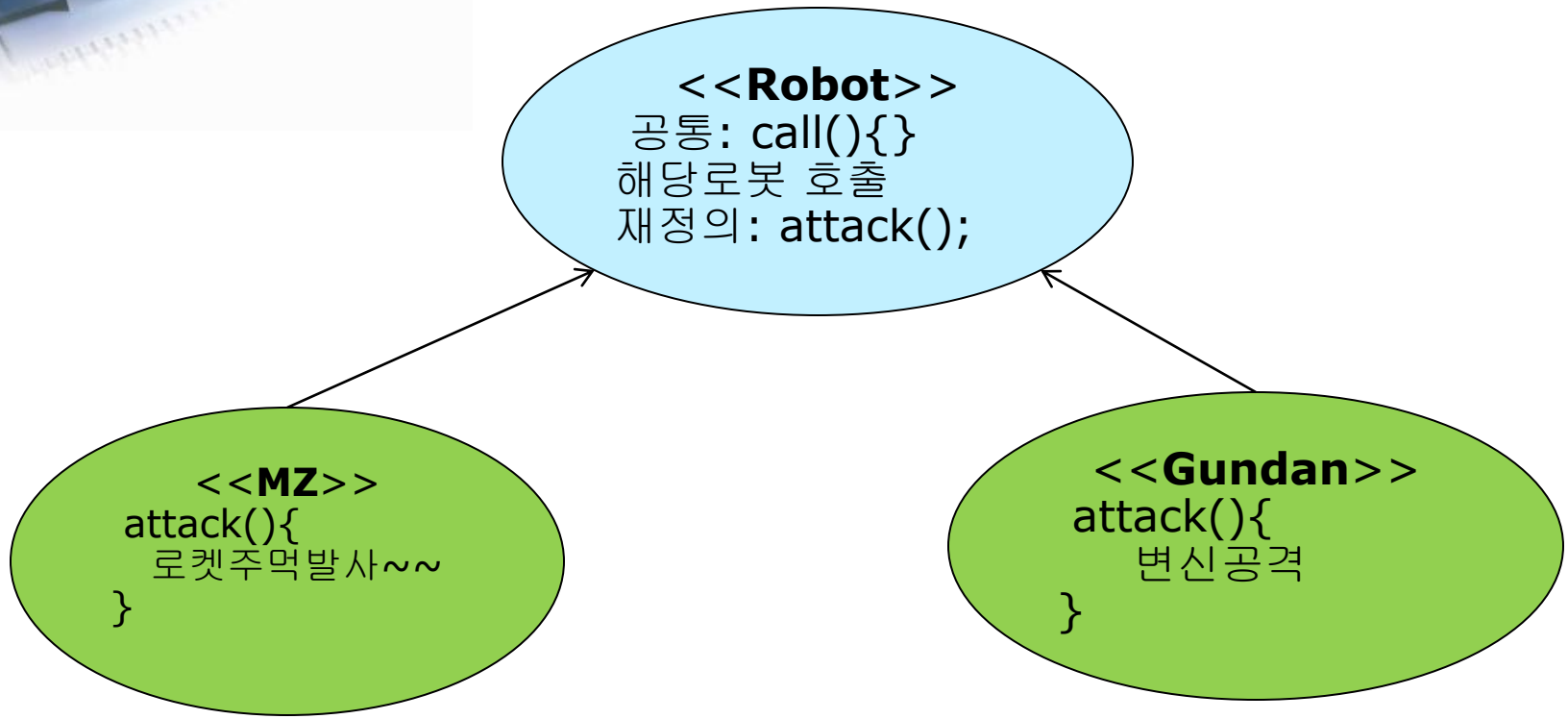
추상 클래스 기본예제

```
class Insect extends Animal01{
    public void move(){
      System.out.println("기어 다니다!!");
    }
}
class Fish extends Animal01{
    public void move() {
      System.out.println("물에서 swimming!!!");
    }
}
```

추상클래스 호출

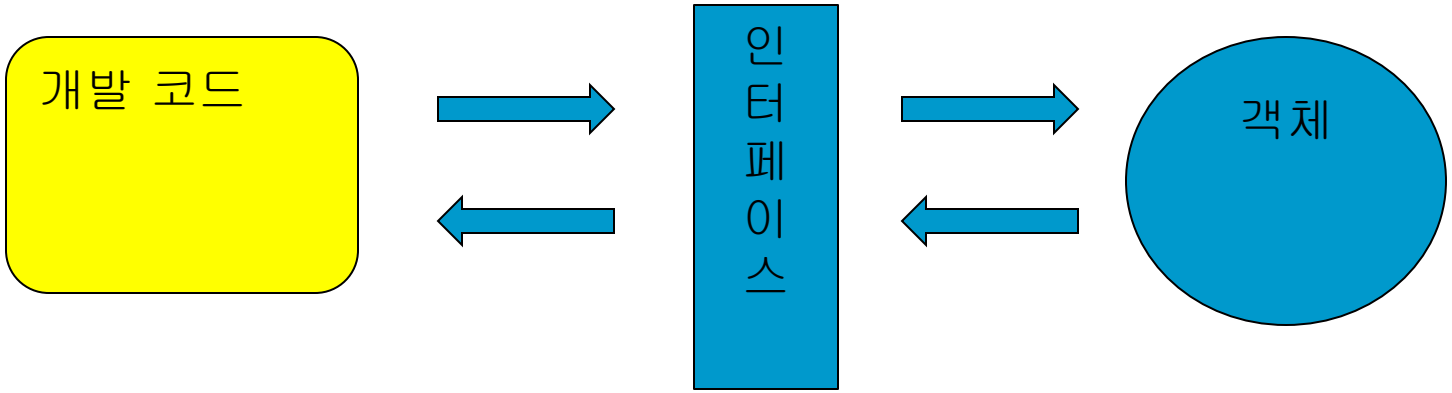
```
public static void main(String[] args) {
    Animal01[] ans={new Bird(), new Insect(),
       new Fish()};
    for(Animal01 an:ans){
    an.eat();
    an.move();
    }
    }
```

확인예제


<<Robot>>
공통: call(){}
해당로봇 호출
재정의: attack();
<<MZ>>
attack(){
로켓주먹발사~~
}
<<Gundan>>
attack(){
변신공격
}

인터페이스

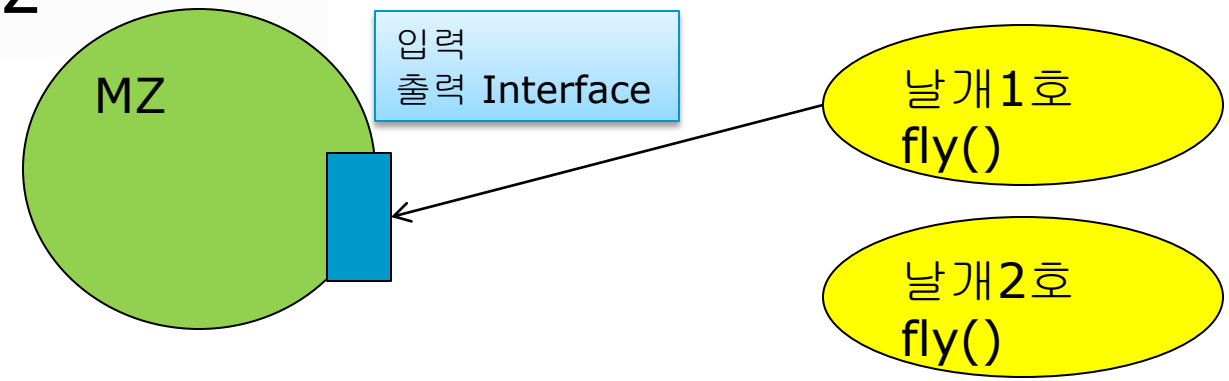
- 객체의 사용 방법을 정의한 type을 말한다.



- 개발 코드 측면에서 코드 변경 없이 실행 내용과 리턴값을 다양화할 수 있는 장점을 가짐

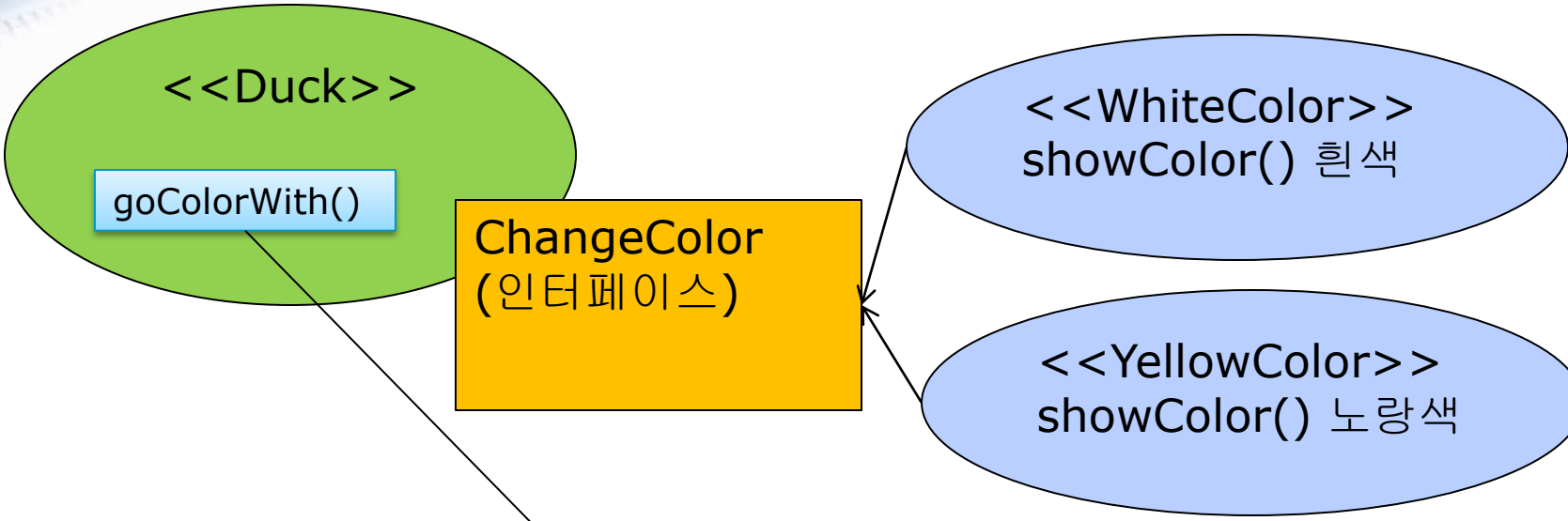
인터페이스 story!!

- Mz



- 날개 X
- interface 는 실제내용을 overriding 할 수 있는 추상 메서드만 구현하고, 실제 클래스들을 차후에 구현하게 확장성을 강력하게 만들 수 있다.

인터페이스 확인예제



흰색 오리가 지나간다.
노랑색 오리가 지나간다.

인터페이스의 멤버들~~

- interface 인터페이스명{
  - 상수
    - [public static final]type 상수명 = 값;
    - ex) int MAX_VOLUME=10;
  - 추상 메서드
    - type 메소드명(매개변수.....);
  - 디폴드 메서드
    - // java 8 부터 지원
    - default 타입 메소드명(매개변수){ 구현내용 }
    - ex) default void show(){
    - System.out.println("안녕하세요");}
  - 정적 메서드
    - static 타입 메소드명(매개변수){ 구현내용 }
    - ex) static void staticShow(){
      System.out.println("좋은 아침");}

기본예제

```
// 인터페이스 선언
interface RemoteControl{
// 필드 ==> 상수 [public static final] 타입
   상수명 = 값;
// 상수는 일반적으로 대문자, _(언더바) 로 구성
int MAX_VOLUME=10;
int MIN_VOLUME=0;
}
```

확인예제

- 1단계
  - Audio 클래스 구현
  - main() 호출
- 숙제(2단계)
  - 인터페이스
    - AttackWay
      - 추상메서드 attack()
  - RocketAttack : AttackWay를 상속받은 실제 클래스
    - attack() : 로켓공격하다.
  - Robot클래스
    - 필드:AttackWay
    - 메서드 showAttack(){
      - attack();
    - 메서드 setAttackWay(AttackWay attway)

익명 구현 객체 만들기.

- 인터페이스를 implements한 클래스는 주로 재사용을 하기 위해 구현을 한다. 그런데, 일회성으로 구현 객체를 만들기 위해 호출과 동시에 만드는 것을 **익명 구현 객체** 라고 한다.
- 형식
  - 인터페이스 변수 = new 인터페이스(){
  - // 인터페이스에 선언한 추상 메소드와 실제 메서드 선언
  - **};**
- **example (main(), 호출하는 클래스)**
  - interface RemoteController{ ,,,, }
  - RemoteController rc01 = new RemoteController(){
    - public void turnOn(){ ...}
    - public void turnOff(){ ...}
  - **};**
  - **rc01.turnOn();**

annonymous interface 활용

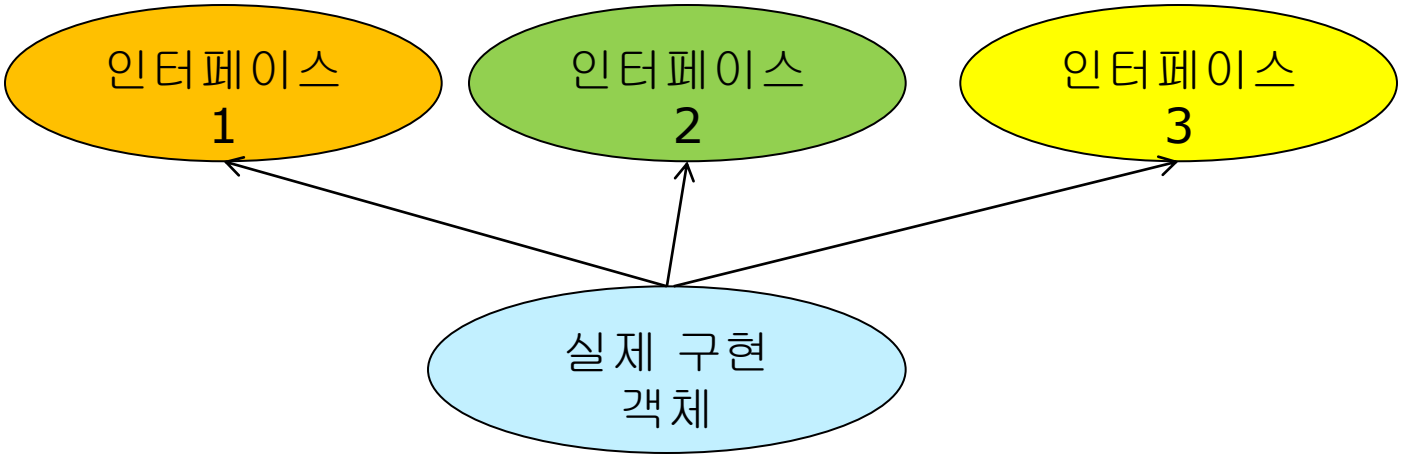
- 자판기 (인터페이스) <<1단계 숙제>>
  - 추상메서드
    - display(); @@@@ 을 판매하는 자판기
  - 공통메서드
    - insertCoin(입력값) @@@ 원이 입력되었습니다.

익명구현 객체 활용

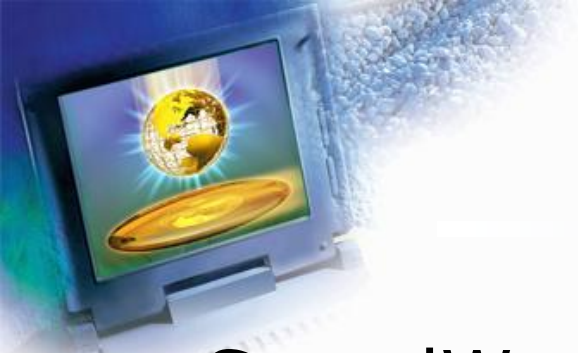
- 자판기 (인터페이스) <<2단계 숙제>>
  - 추상메서드
    - display(); @@@@ 을 판매하는 자판기
    - menu(); 1. @@: @@@원
    - 2. @@: @@@
  - 공통메서드
    - insertCoin(입력값) @@@ 원이 입력되었습니다.
    - choiceMenu(번호나 문자열)
    - outProduct() @@@이 나옵니다.
    - 잔액은 @@@ 입니다.

다중 인터페이스 구현

- 구현클래스는 다중 인터페이스 구현을 할 수 있다.



- public class 구현클래스명 implements 인터페이스1, 인터페이스2, 인터페이스3{
  - 추상메서드 재정의 처리
- }



다중 인터페이스 구현

- SoundWay 인터페이스
  - sound();
- FlyWay 인터페이스
  - flying();
- ShowColor 인터페이스
  - paint();
- 실제 구현 클래스 Duck implements했을 때, 처리할 내용을 구현..

중첩 클래스와 중첩 인터페이스

- 여러 클래스 관계를 통해서 프로그램을 처리하는데, 클래스 내부에 클래스를 선언하는 경우가 있다. 이른 중첩클래스라고 한다.
  - class 클래스이름{
  - class 내부클래스이름{
  - }
  - }
  - 장점 : 두 클래스간의 멤버들을 서로 쉽게 접근할 수 있고, 외부에서 불필요한 관계클래스를 감추어 주는 역할을 한다.
- 인터페이스도 클래스 내부에 선언할 수 있다. 중첩 인터페이스라고 한다.
  - class 클래스이름{
    - interface 내부인터페이스

중첩 클래스 종류

- 멤버 클래스(외부$내부.class)
  - 인스턴스 멤버 클래스 : 외부 객체를 생성해야만 사용할 수 있는 내부 중첩 클래스
    - class 외부{ class 내부{ } }
  - 정적(static) 멤버 클래스 : 외부 클래스로 바로 접근할 수 있는 내부 중첩 클래스
    - class 외부{ static class 내부{} }
- 로컬 클래스 : 메서드가 실행할 때만 사용할 수 있는 중첩클래스(외부$1내부.class)
  - class 외부{
    - void 메서드(){
    - class 내부로컬클래스{}
    - }
  - }

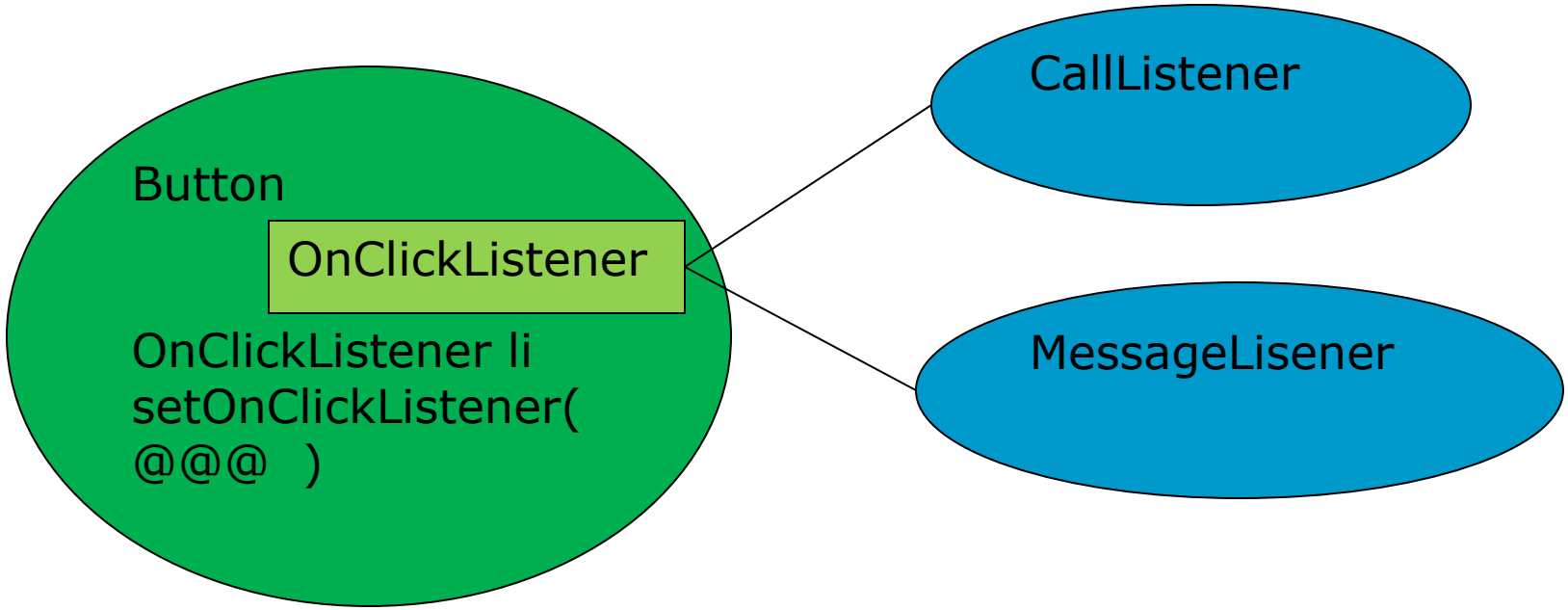
멤버 클래스

- 인스턴스 멤버 클래스와 정적 멤버 클래스의 차이점
  - 외부객체 생성 후
  - static 멤버의 선언가능 여부
    - 인스턴스 멤버 클래스 내부에서는 X
    - 정적 멤버 클래스 내부에서는 O

중첩 인터페이스 예제

- class 클래스명{
  - interface 인터페이스명{
    - void 메서드명(); // 추상 메서드..
  - }
- }

- public class View{
  - public interface OnClickListener{
    - public void onClick(View v);
  - }
- }


Button
OnClickListener
OnClickListener li
setOnClickListener(
@@@)
CallListener
MessageLisener

중첩 인터페이스 확인예제

- (1단계)
  - MusicStartListener : 음악이 켜지는 리스너 구현.
- (2단계)
  - Draw
    - 필드선언..
    - OnTouchListener(중첩인터페이스)
      - onTouch(); --추상메서드..
    - drawing()
  - CircleListener : 동그라미 그려주는 실제클래스
  - TriangleListener : 세모를 그려주는 실제클래스.

정리 및 확인하기

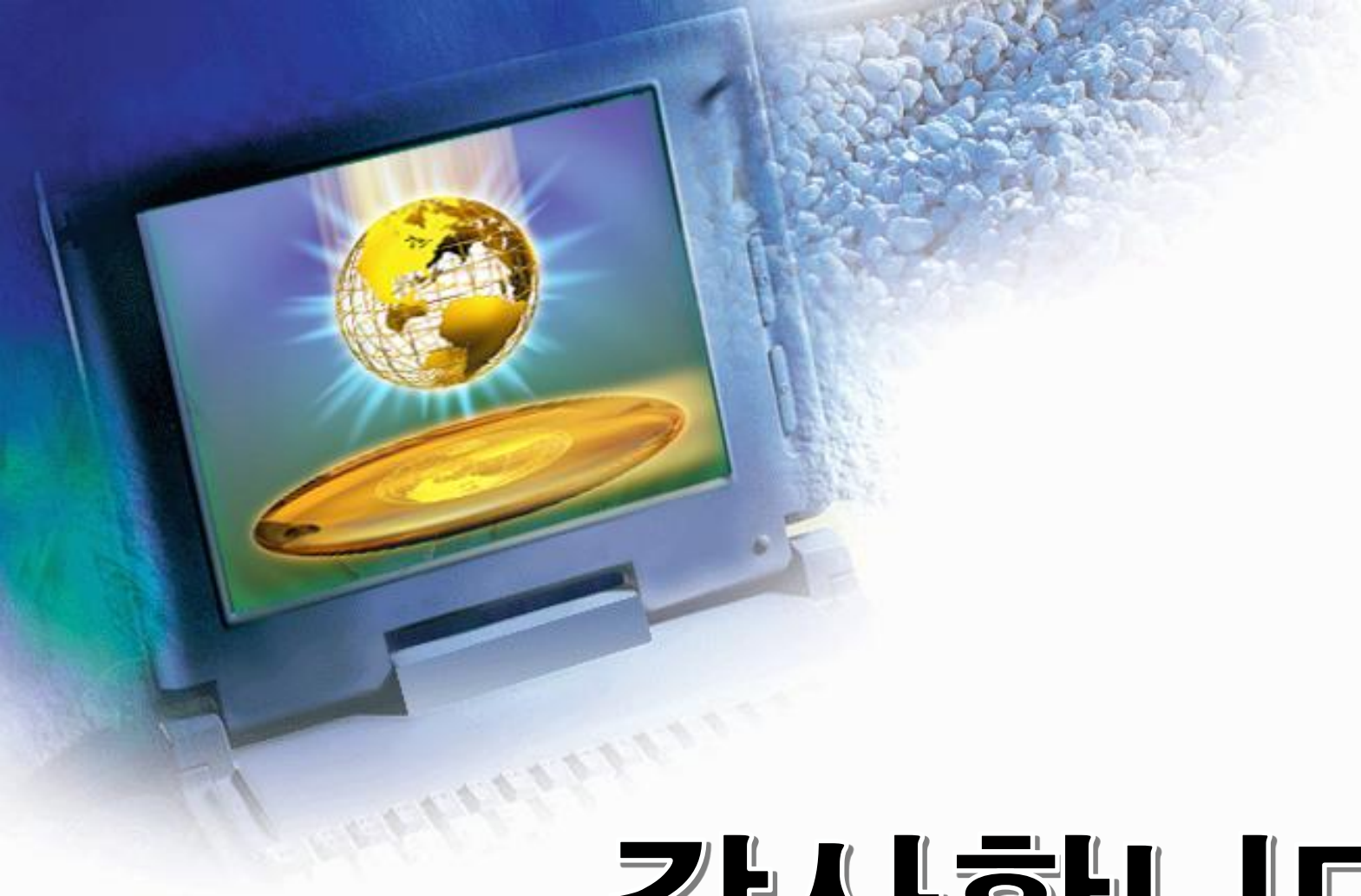
- 자바의 상속에 대한 설명 중 틀린 것은 무엇입니까?
  1. 자바는 다중 상속을 허용한다.
  2. 부모의 메소드를 자식 클래스에서 재정의 할 수 있다.
  3. 부모의 private 접근 제한을 갖는 필드와 메소드는 상속의 대상이 아니다.
  4. final 클래스는 상속할 수 없고, final 메소드는 오버라이딩 할 수 없다.
- 오버라이딩에 대한 설명으로 틀린 것은?
  1. 부모 메소드의 시그니쳐(타입, 메소드명, 매개변수)와 동일해야 한다.
  2. 부모 메소드보다 좁은 접근 제한자를 붙일 수 없다.(public-부모, private – 자식)
  3. protected 접근 제한을 갖는 메소드는 다른 패키지의 자식 클래스에서 재정의 할 수 없다.

정리 및 확인하기

- final 키워드에 대한 설명으로 틀린 것은?
  1. final 클래스는 부모 클래스로 사용할 수 있다.
  2. final 필드는 값이 저장된 후에는 변경할 수 없다.
  3. final 메소드는 재정의(오버라이딩)할 수 없다.
  4. static final필드는 상수를 말한다.
- 인터페이스에 대한 설명으로 틀린 것은 무엇입니까?
  1. 인터페이스는 객체 사용 설명서 역할을 한다.
  2. 구현 클래스가 인터페이스의 추상 메소드에 대한 실제 메소드를 가지고 있지 않으며 추상 클래스가 된다.
  3. 인터페이스는 인스턴스 필드를 가질 수 있다.
  4. 구현 객체는 인터페이스 타입으로 자동 변환된다.

정리 및 확인하기

- 중첩 멤버 클래스에 대한 설명으로 틀린 것은 무엇입니까?
  1. 인스턴스 멤버 클래스는 바깥 클래스의 객체가 있어야 사용될 수 있다.
  2. 정적 멤버 클래스는 바깥 클래스의 객체가 없어도 사용될 수 있다.
  3. 인스턴스 멤버 클래스 내부에는 바깥 클래스의 모든 필드와 메소드를 사용할 수 있다.
  4. 정적 멤버 클래스 내부에는 바깥 클래스의 인스턴스 필드를 사용할 수 있다.



감사합니다 !