BASICS OF JAVA

Java Tutorial #1

신용호

컴퓨터과학과 12

Contents?

1. Java?
2. 구성(Composition)
3. 상속(Inheritance): 클래스의 확장
4. 추상화(Abstraction): 자식에게 넘기기
5. 인터페이스(Interface): 클래스의 틀
6. 묻고 답하기

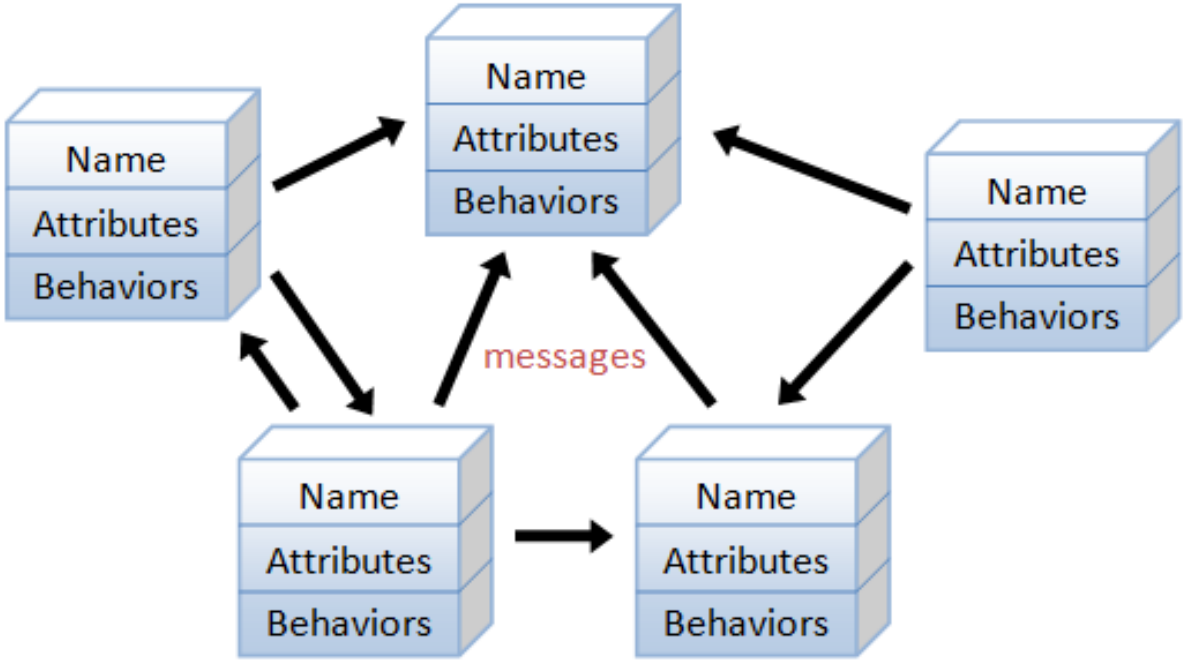
JAVA?

괜찮은 객체 지향 언어

들어가기 전에!

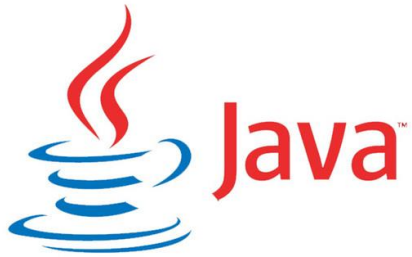
- 이 슬라이드는 기본적인 객체 지향적 패러다임에 대한 이해를 전제로 합니다.
- 때문에 보통 대개 아시는 C/C++과 Java가 다른 점을 위주로 설명합니다.
- 객체 지향 프로그래밍에 대해서 1도 모르시면
  1학년 2학기때 수강했을 객체 지향 프로그래밍 과목을 다시 봐주세요.
  ~~*절대 귀찮아서 입니다.*~~

객체 지향 프로그래밍



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

Java



- 한 잔의 커피?
- Sun Microsystems의 James Gosling 외 여러 개발진이 1995년에 만든 객체 지향적 프로그래밍 언어
- 2016년 현재 가장 많이 쓰이는 언어
  Reference: http://www.tiobe.com/tiobe-index//
- 특징: Object-Oriented, Class-Based, Concurrent, Few Implementation Dependencies, Write Once Run Everywhere, Bytecode and JVM, …

그런 거 잘 모르겠고…

Java VS C++

- C++은 C와의 호환성 때문에 뭔가 어정쩡한 객체 지향적 언어
  하지만 Java는 그런 것 없어서 좀 더 객체 지향적이에요.
- 하지만 Java의 태생이 UNIX/Linux라서 C/C++과 Syntax는 비슷합니다!
- C/C++에서 여러분을 고생시킨 포인터, 참조형(Reference) 등
  Java에서는 싹 사라졌어요!
- 메모리 할당을 하고 메모리를 다시 풀어줘야 하는 C/C++
  Java에서는 알아서 참조하지 않는 메모리를 회수한답니다.
  (Garbage Collector)
- C++에서는 main함수가 어디에든 위치해도 되지만,
  Java는 모든 것이 클래스로 이루어져야 한답니다.
  심지어 main함수도 클래스 안에 존재해요!

백문이 불여일견!

- 이 외에도 여러 특징이 있어요.
- 하지만 백문이 불여일견이듯
  직접 해보는 게 가장 좋겠지요?
- 이제 Java 코딩하러 들어가 볼까요?

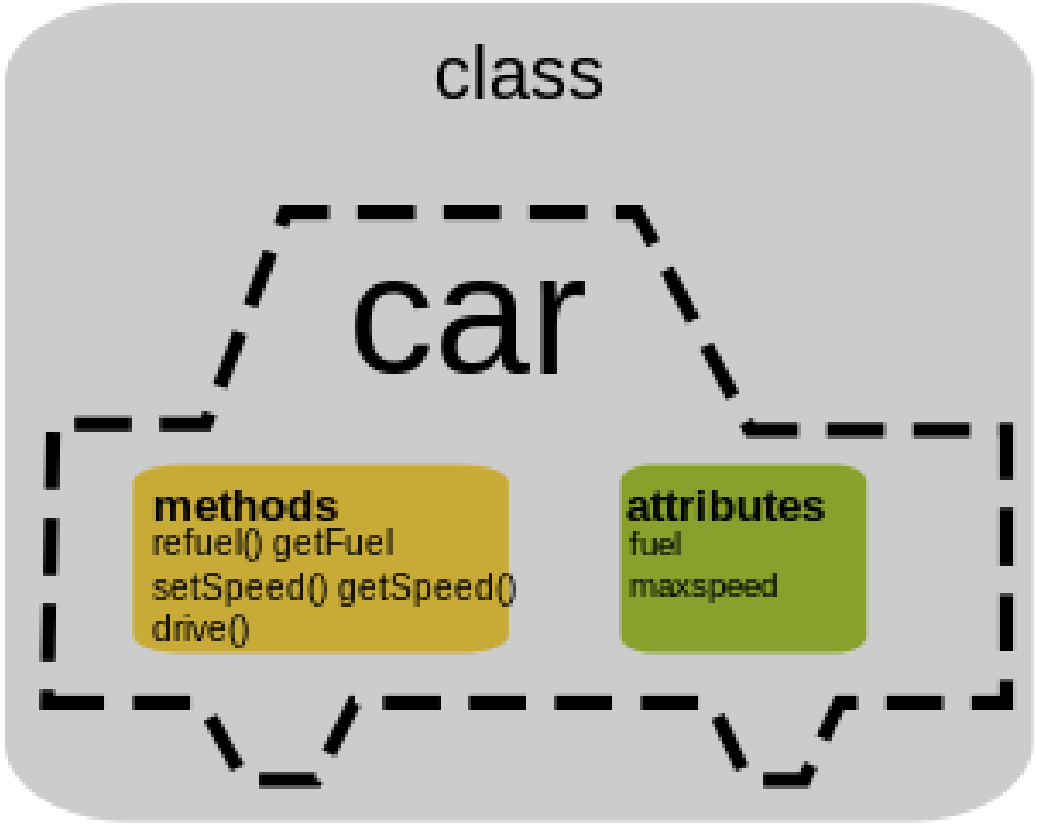
구성

Composition
Java는 무엇으로 이루어져 있을까?

클래스의 구성

- 필드(Field)
  - 클래스의 특징(Property)
  - 속성(Attribute)
  - 멤버 변수(Variable)

- 메소드(Method)
  - 클래스의 활동(Activity)



처음이니 Hello World~

- 시작은 원래 그렇듯이 Hello World~
- Explorer > (오른쪽 마우스 클릭) > New > Project로 새 프로젝트 만들게요.
- 프로젝트 명은 원하는 대로~
- 같은 방법으로 New > Class로 새 클래스를 만들게요~
- 클래스 명은 Zoo로 할게요.
- 오른쪽 코드를 입력하세요~

```
public class Zoo {

	public static void main(String[] args) {

		System.out.println("Hello World!");

	}

}
```

주의할 점!

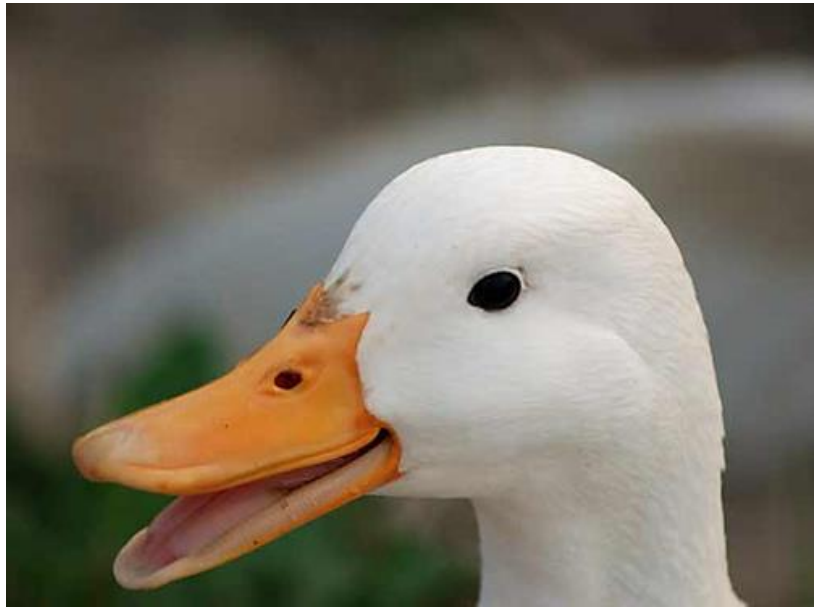
- java 파일 이름과 클래스의 이름은 같아야 합니다!
- 하나의 java 파일에는 하나의 클래스만 정의를 하도록 합시다!
- main 함수는 C/C++과 똑같이 프로그램이 시작하는 함수입니다.
  전체 프로젝트에서 반드시 하나만 있어야 한답니다.

실행을 해봅시다.

- Zoo.java를 현재 창에 두고 오른쪽 클릭!
- Run as > Java Application 클릭!
- 콘솔 창에 Hello World!가 보이시나요?

클래스명이 Zoo? 동물원?

- 동물원에는 동물이 있어야지요!
- 먼저 오리를 넣고 싶습니다.
- 그냥 오리 말고,
  하얀 오리와 까만 오리를 넣고 싶네요~

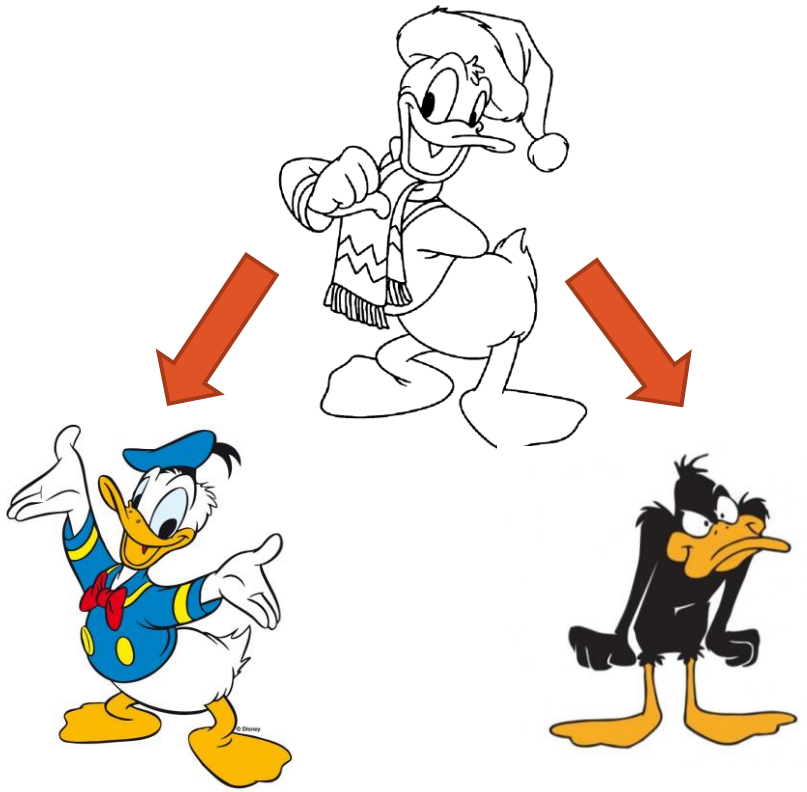


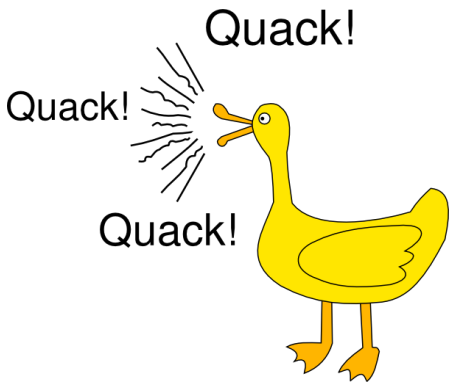
오리를 어떻게 만들까?

- 먼저 오리라는 큰 틀은 똑같죠?
  (클래스)

- 하얀 오리와 까만 오리를 만들고 싶으니까,
  오리의 털 색깔이 오리의 특징적인 부분이겠지요?
  (필드)

- 두 오리 모두 열심히 꽥꽥댑니다.
  두 오리의 울음 소리는 별반 다르지 않겠죠?
  (메소드)





다음을 작성해 보세요~

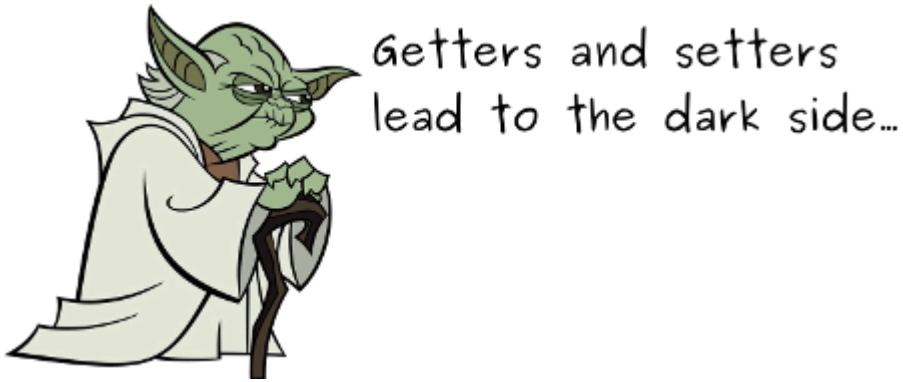
오리는 클래스!

```
public class Duck {

	private String furColor;

	public void quack() {
		System.out.println("Quack! Quack!");
	}

}
```

털 색깔은 필드!

꽥꽥거리기는 메소드!

이런! 털 색깔이…



- 오리의 털 색깔, furColor가 private이네요?
  이럴 때 필요한 것은 무엇일까요?

- 바로 getter와 setter 함수죠!
  직접 입력하시려고요? 그러지 말고, Eclipse의 힘을 믿으세요.

- 오른쪽 클릭 > Source > Generate Getters and Setters

- Generate Constructors using Fields를 사용하면
  생성자를 만들 때도 편하답니다.

다음 부분이 추가되었어요~

```
public Duck(String furColor) {
      this.furColor = furColor;
}
```

새로운 생성자

```
public String getFurColor() {
      return this.furColor;
}

public void setFurColor(String furColor) {
      this.furColor = furColor;
}
```

새로운 Getter와 Setter

이 부분은 Check!~

- 생성자를 정의하지 않으면 아무 변수도 받지 않는 기본 생성자가 생겨요.
  하지만 생성자를 하나라도 정의하면 기본 생성자를 직접 구현해야 한답니다.
- Getter와 Setter를 활용하는 것은 기본 중의 기본!
  하지만 귀찮죠. 그러니 Eclipse의 자동 완성 기능을 활용해 보세요.
  다른 여러가지 자동 완성 기능이 여러분의 코드 속도를 높여준답니다.
- 클래스 이름은 대문자로 시작하고, (Duck)
  필드 이름, 메소드 이름은 소문자로 시작합니다.
  필드 이름은 대개 명사형이고요. (furColor)
  메소드 이름은 서술어로 이루어 진답니다. (getFurColor, quack)
- 이외에도 여러가지 관습들이 있으니 잘 확인해 주세요!

동물원에 오리들이~

```
public class Zoo {

	public static void main(String[] args) {

		Duck blackDuck = new Duck("Black");
		Duck whiteDuck = new Duck("White");

		blackDuck.quack();
		whiteDuck.quack();

	}

}
```

오리만 있으면 심심하죠!

- 이제는 우리의 상징! 독수리를 데려오고자 합니다.
- 독수리는 머리 색깔이 독특한데요…
  초록 머리 독수리와 푸른 머리 독수리가 있어요.
- 그리고 독수리들은 이렇게 운다죠…
  AKARAKA~

독수리야 어디 있니?

```java
public class Eagle {

	private String headColor;

	public Eagle(String headColor) {
		this.headColor = headColor;
	}


	public String getHeadColor() {
		return headColor;
	}

	public void setHeadColor(String headColor) {
		this.headColor = headColor;
	}


	public void yell() {
		System.out.println("Akaraka!");
	}

}
```

```java
Eagle greenHeadedEagle = new Eagle("Green");
Eagle blueHeadedEagle = new Eagle("Blue");

greenHeadedEagle.yell();
blueHeadedEagle.yell();
```

어디에 입력해야 할까요?

우리 동물들이 굶어 죽어가고 있어요!

- 동물들에게 음식과 물을 주어야 겠어요.
- 먼저 음식과 물 각각의 객체가 필요하겠죠?
- 아무 것도 안에 없는건…
  여러분이 구현해 보세요~
  ~~*그냥 더 만들기 귀찮았어요.*~~

Food.java를 만들고!

```
public class Food {

}
```

Water.java를 만들고!

```
public class Water {

}
```

자 이제 음식과 물을 먹어야지요?

- 먹거나 마시는 건 행동이니까 메소드로 구현을 해야겠지요?

- 근데 음식을 먹는 거나 물을 마시는 거나 비슷하지 않나요?
  그러면 메소드 이름을 비슷하게 하는 게 좋겠죠?
  아니! 기왕이면 똑같은 게 좋을 거예요!

- 여기서 오버로딩(Overloading)을 활용하겠습니다!
  ~~*이거 모르면 컴프, 객지프부터 다시 시작…*~~

메소드 이름이 똑같네?

비슷한 로직은 비슷한 이름을,
기왕이면 똑같은 이름이 좋겠죠?

```java
public void eat(Food food){
    System.out.println("Eat Food");
}

public void eat(Water water) {
    System.out.println("Drink Water");
}
```

파라미터의 구성이 다르면
같은 메소드 이름을 활용할 수 있어요!
(오버로딩)

오리도 살아야 하고, 독수리도 살아야 하니까
둘 다 모두 구현해 주어야 겠지요?

자! 잘 동작 하나요?

```
Food food = new Food();
Water water = new Water();

blackDuck.eat(food);
whiteDuck.eat(water);


greenHeadedEagle.eat(food);
blueHeadedEagle.eat(water);
```

짝짝짝! 잘하셨어요! 근데…

- 오리 코드와 독수리 코드에 먹고 마시는 행동이 완전히 똑같죠?

- 만약에 eat 함수에 수정을 해야한다면?
  오리 코드도 봐야하고 독수리 코드도 봐야합니다.
  매우 번거롭겠지요?

- 그럼 어떻게 하면 될까요?

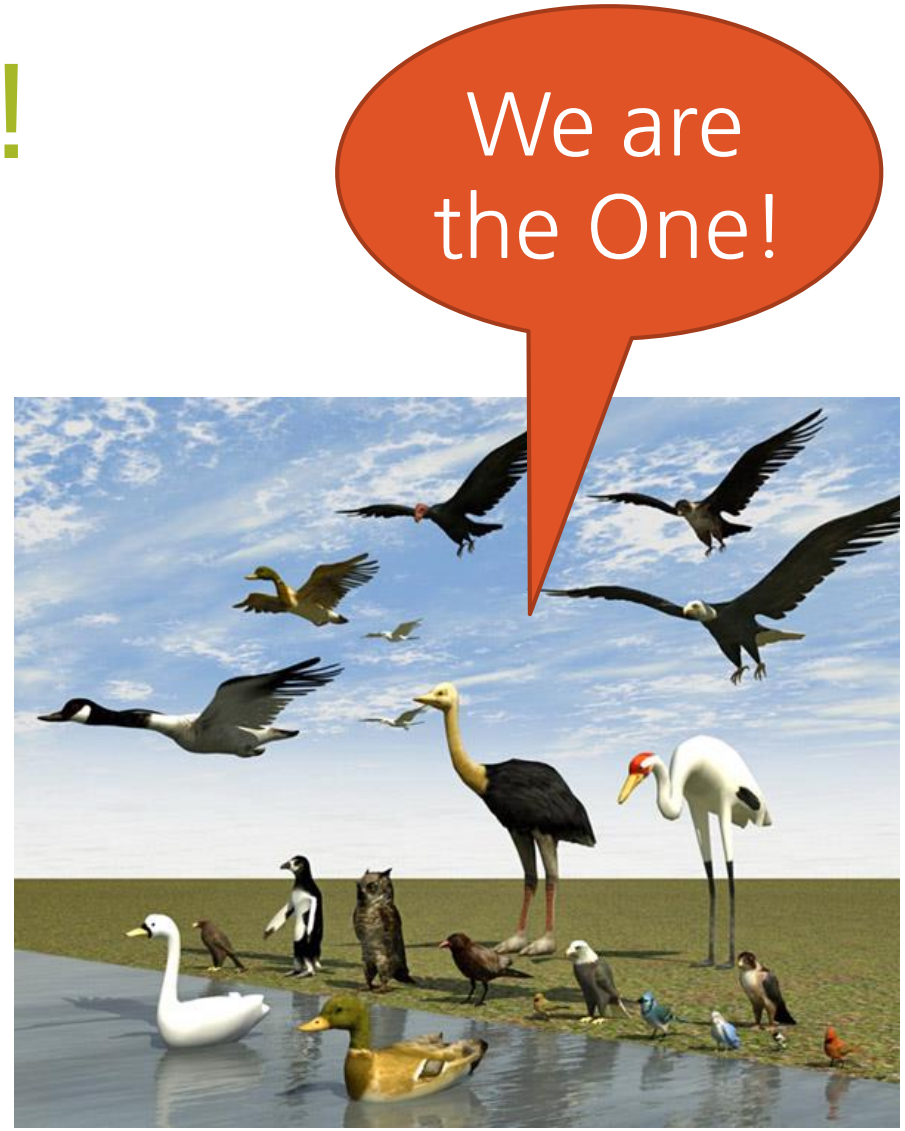
상속

Inheritance
클래스의 확장

그래! 우리는 새(Bird)였어!

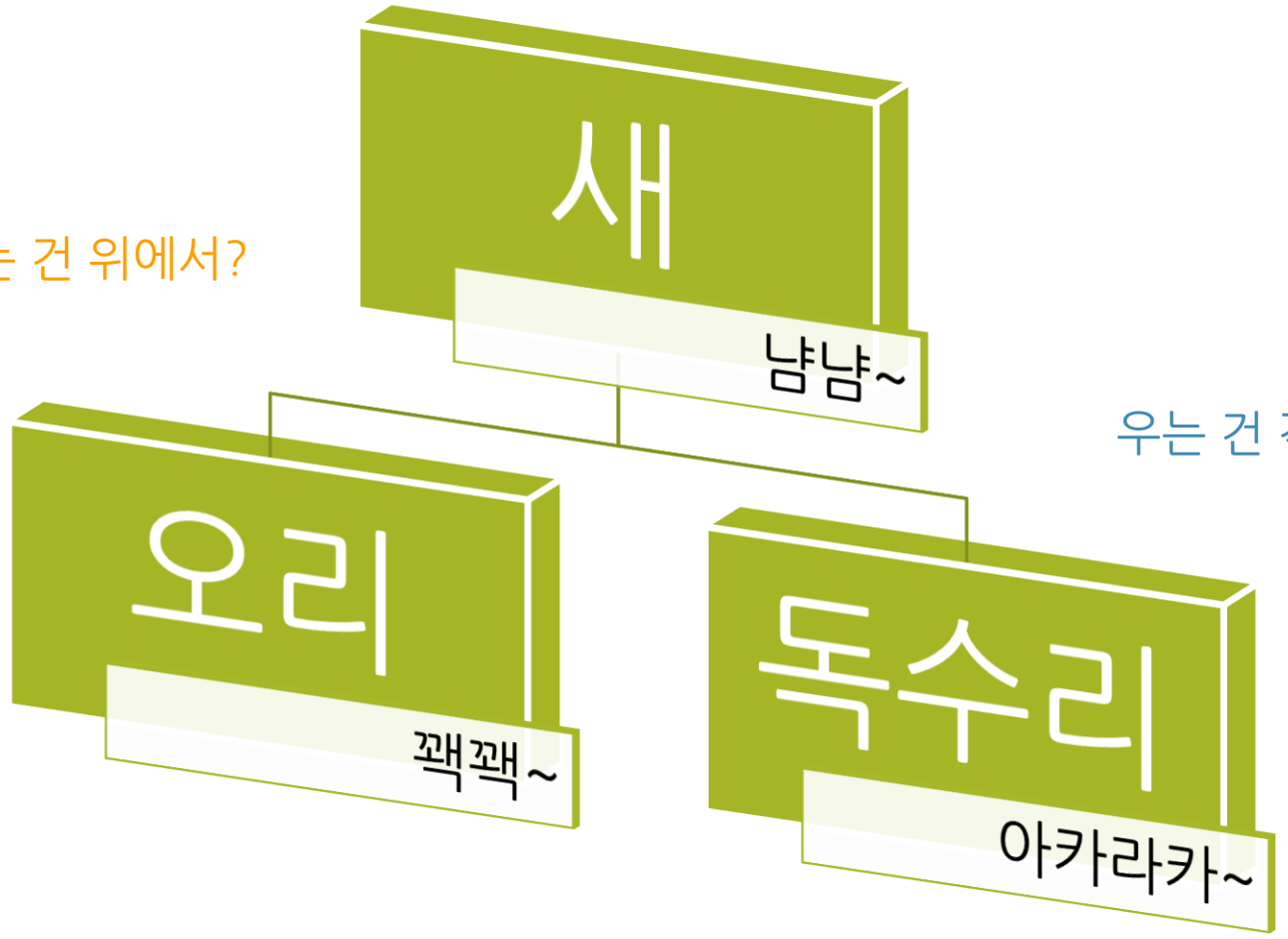
- 오리 : 동물계 **조류** 기러기목 오리과
  독수리 : 동물계 **조류** 매목 수리과

- 둘을 아우르는 더 큰 객체를 생각할 수 있을까요?



그러니까 이런 느낌?

밥 먹고 물 마시는 건 위에서?



우는 건 각자 따로 따로?

새를 구현합시다!

- 이제 오리와 독수리의 상위의 객체인 새를 구현합시다.
- 새 객체에서는 밥 먹고 물 마시는 것을 구현할 겁니다.
- 먼저 새 객체를 다음과 같이 생성해 주세요!

```
public class Bird {

	public void eat(Food food) {
		System.out.println("Eat Food");
	}

	public void eat(Water water) {
		System.out.println("Drink Water");
	}

}
```

오리와 독수리는?

```java
public class Duck extends Bird {

	private String furColor;


	public Duck(String furColor) {
		this.furColor = furColor;
	}


	public String getFurColor() {
		return this.furColor;
	}

	public void setFurColor(String furColor) {
		this.furColor = furColor;
	}


	public void quack() {
		System.out.println("Quack! Quack!");
	}


}
```

eat() 함수가 없는 것도 Check!

```java
public class Eagle extends Bird {

	private String headColor;


	public Eagle(String headColor) {
		this.headColor = headColor;
	}


	public String getHeadColor() {
		return headColor;
	}

	public void setHeadColor(String headColor) {
		this.headColor = headColor;
	}


	public void yell() {
		System.out.println("Akaraka!");
	}


}
```

Zoo는 그대로?

- Zoo는 별다른 코드의 변화를 주지 않아도 됩니다.

- 원래 오리와 독수리 각각에서 구현되어 있던 것을
  새라는 상위의 객체로 묶어냈기 때문이지요.

- 자 질문! Zoo에서 다음과 같이 바꾼다면 이는 동작을 할까요?

```
Bird blackDuck = new Duck("Black");
Bird whiteDuck = new Duck("White");

Bird greenHeadedEagle = new Eagle("Green");
Bird blueHeadedEagle = new Eagle("Blue");
```

이것이 상속입니다!

- 상속을 하면 상위의 객체의 모든 것들을 물려받게 됩니다.
  필드, 메소드 상관 없이 말이죠!
- 객체를 적절하게 엮어내는 것은 OOP에서 매우 중요한 일입니다.
  공통되는 부분이 있다면 상위의 객체로 묶어내어 관리하는 것이 편하죠
- 곰곰이 생각해 보니, 오리가 꽥꽥 우는 것하고 독수리가 **우렁차게** 우는 것하고
  그 원리는 비슷하다는 생각이 드네요. 대신 내는 소리가 다르지만요.
- 여기서는 무엇을 사용하면 좋을까요?
  잘 생각해 보세요!

추상화

Abstraction

자식에게 넘기기

정답은?

- 정답은 바로! 오버라이드(Override)입니다.
- 오버라이드는 부모 객체에서 구현되었거나 정의된 메소드를 자식이 다시 정의하는 것입니다.
- 그렇게 하면 동일한 메소드 이름으로 오리와 독수리가 서로 다른 소리를 낼 수 있답니다!

새(Bird)를 수정해 볼까요?

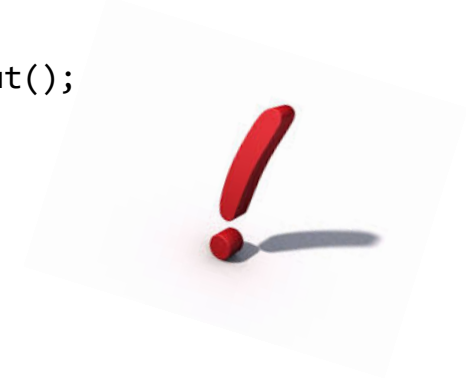
```
public class Bird  {

	public void eat(Food food) {
		System.out.println("Eat Food");
	}

	public void eat(Water water) {
		System.out.println("Drink Water");
	}

	public void shout() {

	}
}
```

- 새가 우는 함수를 shout()라고 합시다.
- 이 shout() 안에는 음…
  무엇을 적어 넣어야 할까요?
- 차라리 아무 것도 적지 않는 건 어떤가요?
  새(Bird) 객체가 구체적인 것을 가리키는 것이 아닌 것처럼
  shout()도 무언가 추상적인 게 아닐까요?

차라리 없애버려!

- shout() 앞에 abstract가 붙고, 괄호 대신 세미콜론이 붙었습니다.
- 이는 추상 메소드를 정의하는 방법입니다. 추상 메소드는 현재 객체에서는 별 내용이 없으니 자손 객체에서 알아서 구현해서 사용하라는 뜻이에요.
- 추상 메소드를 하나라도 포함하고 있으면 해당 클래스는 추상 클래스가 된답니다. 그래서 Bird 앞에도 abstract가 붙었어요!

```
public abstract class Bird {

	public void eat(Food food) {
		System.out.println("Eat Food");
	}

	public void eat(Water water) {
		System.out.println("Drink Water");
	}

	public abstract void shout();
}
```



잠깐! 직접 입력하기 전에!

- 새(Bird)가 추상 클래스가 되면서
  오리와 독수리한테 무엇인가 잘못되었다는
  신호가 오네요?
- 직접 shout()를 치지 말고 Eclipse의 또 다른
  놀라운 기능을 활용해 봅시다!
- 클래스 이름 위에 마우스를 올려다 놓으세요.
  Add unimplemented method 클릭!
- 오른쪽의 코드가 생성된 것이 보이나요?
- 자 이제 새로 생성된 메소드를 구현하고 필요
  없어진 메소드는 지워버려요!

```
@Override
public void shout() {
	// TODO Auto-generated method stub

}
```

동물원도 조금 바뀌겠죠?

```
blackDuck.quack();
whiteDuck.quack();


greenHeadedEagle.yell();
blueHeadedEagle.yell();
```

```
blackDuck.shout();
whiteDuck.shout();

greenHeadedEagle.shout();
greenHeadedEagle.shout();
```

꽤 괜찮은 동물원이 되었어요!

- 음 근데 너무 새들만 있는 것 같네요.
  요즘 중요한 것은 다양성인데 말이죠!

- 정했어요!
  물고기를 넣어야 겠습니다.

- 물고기는 새와 완전히 다르니까
  완전히 다른 객체를 추가해야겠지요?
  근데 과연 완전히 다를까요?

인터페이스

Interface

클래스의 틀

물고기를 만들자!

```
public class Fish {

	public void swim() {
		System.out.println("Splash! Splash");
	}

}
```

잠깐! 오리는요?

- 오리도 헤엄을 칠 수 있어요!
- 그럼 오리와 물고기의 부모 객체를 생성해서 상속을 해야할까요?
- 먼저 그런 기괴한 조합은 상상도 하기 힘드네요.
- 그보다, 과연 다음 코드가 가능할까요?

```
public class Duck extends Bird, Fish
```

슬프지만…

- 슬프게도 Java에서는 다중 상속을 지원하지 않는답니다.
- 그럼 어쩔 수 없이 swim() 함수를 생성해야 하는 걸까요?
- 언제나 길은 있습니다.
  우리에게는 인터페이스가 있습니다.

인터페이스가 뭐야?

- 직관적으로 체크리스트라고 생각하면 편해요.
  즉 인터페이스를 구현하는 객체는 해당 인터페이스가 하라고 명시한 체크리스트를 모두 해야한답니다.

- 좀 더 엄밀하게는 추상 메소드로만 이루어진 객체입니다.
  그래서 이를 구현한 객체는 해당 추상 메소드를 모두 구현해야 하는 것이죠.

돌아와서…

```
public interface Swimmable {
	public void swim();
}
```

- 인터페이스의 이름은 대개 -able로 끝나는 경우가 많아요

```
public class Duck extends Bird implements Swimmable
```

```
public class Fish implements Swimmable
```

- Eclipse의 자동 완성 기능을 활용해서 unimplemented method를 채워넣어 보세요!

다시 동물원으로~

```
Fish fish = new Fish();
fish.swim();

blackDuck.swim();
```

결과가 어떤가요?

멋진 동물원이네요!

- 우리의 동물원에 이제 하얀 털 오리, 까만 털 오리, 파란 머리 독수리, 초록 머리 독수리, 심지어 물고기도 있어요!

- 하지만 아직 만족하긴 이르죠.
  더욱 멋진 동물들로 가득 채워 넣는 것은 어떨까요?

연습문제

1. 토끼와 사자에 해당하는 객체를 구현해 보세요.
2. 포유류에 해당하는 객체를 구현하고 토끼와 사자에게 상속하세요.
3. 토끼와 사자 모두 음식을 먹고 물을 마시죠. 조류와 포유류 모두를 아우르는 동물이라는 최상위 객체를 만들고, 여기에 해당 메소드를 옮겨 보세요.
4. 아직 새들은 날지를 못하고, 포유류들은 걷지를 못하네요. (ㅜㅜ)
   하지만 새만 날아다니라는 법도 없고, 포유류만 걸어다니라는 법도 없지요.
   이를 인터페이스를 활용하여 각각의 객체를 구현해 보세요.
5. 부모 객체인 새(Bird)를 통해 자식 객체인 오리와 독수리를 받을 수 있었죠.
   그러면 과연 인터페이스를 통해서 자식 객체를 받을 수 있을까요?

수고하셨습니다.

질문이 있으면 말씀해 주세요~